DIRECT SERVO CONTROL OF POSITIONAL DERIVATIVES FOR 5-AXIS CNC MACHINE TOOLS USING DENSELY-SAMPLED TOOLPATHS

A Dissertation
Presented to
The Academic Faculty

By

Roby Lynn

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Mechanical Engineering

Georgia Institute of Technology

May 2019

DIRECT SERVO CONTROL OF POSITIONAL DERIVATIVES FOR 5-AXIS CNC MACHINE TOOLS USING DENSELY-SAMPLED TOOLPATHS

Approved by:

Dr. Thomas Kurfess, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Christopher Saldana
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Cassandra Telenko
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Jarek Rossignac
School of Interactive Computing
Georgia Institute of Technology

Dr. Tommy Tucker
Tucker Innovations, Inc.

Dr. Moneer Helu
Systems Integration Division
National Institute of Standards and Technology

Date Approved: February 12, 2019

ACKNOWLEDGEMENTS

I would first like to thank the National Science Foundation for providing the funding and freedom for me to carry out this research. I owe special thanks to my advisor, Dr. Thomas Kurfess, for his energy, encouragement, and guidance on both personal and professional levels. Thanks are also due to Dr. Tommy Tucker, who provided instrumental guidance and development support necessary to complete this research. And to Dr. Moneer Helu, thank you for being such a fantastic colleague and friend, and thanks for your invaluable standards perspective on this work. And to my other committee members, Dr. Chris Saldana, Dr. Jarek Rossignac, and Dr. Cassandra Telenko, thanks for supporting me through all the research efforts I was engaged in while at Georgia Tech, especially this project. Thanks to my lab mates, both junior and senior, for your guidance and companionship. I will always remember the fun times we shared. I grateful to my parents and to Rebecca, all three of whom I love very dearly. Thank you for always supporting me through both good times and bad. Finally, I owe a special thank you to my father, David Lynn, for being such a tremendous inspiration to me for my entire life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Ever-increasing quality and complexity requirements for machined parts have led to the development of computer-numerical control (CNC) machine tools with high numbers of servo axes capable of tightly coordinated motion. These machine tools are usually programmed using computer-aided manufacturing software that creates toolpaths for machining surfaces and features selected by a user. Voxel-based computer-aided manufacturing (CAM) software has shown great potential in both creating machining plans for highly complex parts and performing realistic simulations of material removal that would be impractical with current industrial CAM systems. Voxel models allow for the creation of toolpaths that follow the exact surface of a given part on a voxel-by-voxel basis, which enables the recreation of very fine surface details on a machined part. The created toolpaths are translated by the CAM system into a format readable by the machine, known as G-Code, which consists of points and maximum velocities that the machine should follow in order to trace out the desired path. For toolpaths created from a voxel model, this G-Code program consists of many small linear movements for each axis of the machine tool.

Specifying toolpaths to the machine in G-Code has a number of limitations: first, many commands are machine specific, which causes compatibility issues between the CAM system and the CNC; second, translating a toolpath into G-Code causes a loss of valuable process control data between the CAM system and the CNC; and third, the use of G-Code forces the CNC to spend valuable compute cycles performing online trajectory planning using a worst-case approach that can prevent the cutting tool from reaching its programmed maximum velocity. Even the most sophisticated CNC machine tool control systems are unable to maintain the programmed tool velocity while machining a toolpath created from a complex voxel model. This causes the machine to not execute the exact toolpath provided by the CAM system, which renders offline simulations of machining and material removal less effective. Much research has focused on finding optimal tool velocities to traverse a

path more quickly in order to reduce machining time, but all of these works still rely on G-Code. To overcome the limitations present in G-Code programming, this research develops and evaluates a new solution to offline trajectory planning and control that is enables a CNC machine tool to follow a densely-sampled toolpath (such as one created from a voxel model) at the kinematic limits of each axis. Additionally, the proposed approach will allow for the communication of densely-sampled motion trajectories that would be impossible with standard G-Code.

The contributions of this work are as follows: first, a generalized framework and accompanying control system for direct transmission of dense data to and from the machine tools servo controllers directly from a voxel-based CAM system is developed; second, a reference implementation of this approach is performed on an open-source CNC platform known as Machinekit; third, near-realtime simulation and analysis capabilities from within the CAM system are developed and discussed; and fourth, the accuracy of motion realizable by the new control system is validated using complex toolpaths created from the CAM system and performance is compared to the standard G-Code programming method.

TERMINOLOGY

CNC: Computer Numerical Control

CAD: Computer-Aided Design

CAM: Computer-Aided Manufacturing

MES: Manufacturing Execution System

SCADA: Supervisory Control and Data Acquisition

ERP: Enterprise Resource Planning

PDM: Product Data Management

PLM: Product Lifecycle Management

VM: Virtual Machine

DaaS: Desktop-as-a-Service

IaaS: Infrastructure-as-a-Service

CC: Cloud Computing

OPC-UA: Open Platform Communications Unified Architecture

IoT: Internet of Things

HPC: High-Performance Computing

GPU: Graphics Processing Unit

FKT: Forward Kinematic Transformation

IKT: Inverse Kinematic Transformation

MRR: Material Removal Rate, or the volumetric rate of material removal from the workpiece

DFM: Design for Manufacturability

TP: Trajectory Planner

DoF: Degree(s) of Freedom

WPC: Workpiece Coordinates (also referred to as tool space coordinates)

TCPC: Tool Center Point Control

RTOS: Realtime Operating System

TCP: Transmission Control Protocol

HTTP Hypertext Transfer Protocol

XML: eXtensible Markup Language

FIFO: First-In, First-Out

HAL: Hardware Abstraction Layer

PMSM: Permanent Magnet Synchronous Motor

API: Application Programming Interface

GIL: Global Interpreter Lock

G-Code: The industry standard programming language for CNC machine tools

Block: A single line of G-Code, which typically contains movement commands for each axis

Feedrate: Traversal velocity of the cutting tool along the workpiece surface

Surface Speed: Tangential velocity of cutting tooth due to spindle rotation

Voxel: A small cube used for volumetric discretization of a three-dimensional object

Toolpath: A collection of tool pose frames that define the position and orientation of a cutting tool necessary to machine a given part

Trajectory: The positional derivative progression of the cutting tool or joint actuators along a toolpath

CHAPTER 1
INTRODUCTION AND BACKGROUND

The ever-increasing demand for higher automation and production capacity from machining operations has led to the development of computer numerical control (CNC) machine tools that enable the production of higher quality parts at increased production rates than would be realizable with manual machine tools. Instead of relying on a human operator to turn handwheels to guide the motion of a cutting tool, CNC machines use servomechanisms that are controlled by a programmable computer. The computer is programmed to move a cutting tool along a toolpath, which is the sequence of movements that must be followed to machine a given part. Typically, machine motion is controlled using servomotor-driven screws that can precisely position a cutting tool with accuracy on the order of micrometers.

CNC machine tools can be configured in a multitude of ways, with multiple movement axes that control the motion of the tool or the workpiece. For instance, a common configuration for a milling machine is the 3-axis machine: this machine tool can realize relative motion between the cutting tool and the workpiece in 3-dimensional Cartesian space; this is performed using three separate movement axes that control relative X, Y and Z position between the part and the workpiece. This can be accomplished by any combination of separate axes on the tool, workpiece, or both. For some parts, however, a machine tool that can only perform translational motion may not be sufficient to machine all of the features on the part. As a result, more sophisticated machine tools are equipped with rotary axes in addition to the three translational axes. These rotary axes enable the control of the orientation of a cutting tool, which is the angle that the tool makes with the part surface. In the case of a milling tool, the orientation is the angle between the tool axis vector and the translational axes of the machine, as illustrated in Figure 1.1 [1]. The cutting tool, shown in yellow, can be positioned in three dimensions and oriented using both θ and ϕ angles. The combination

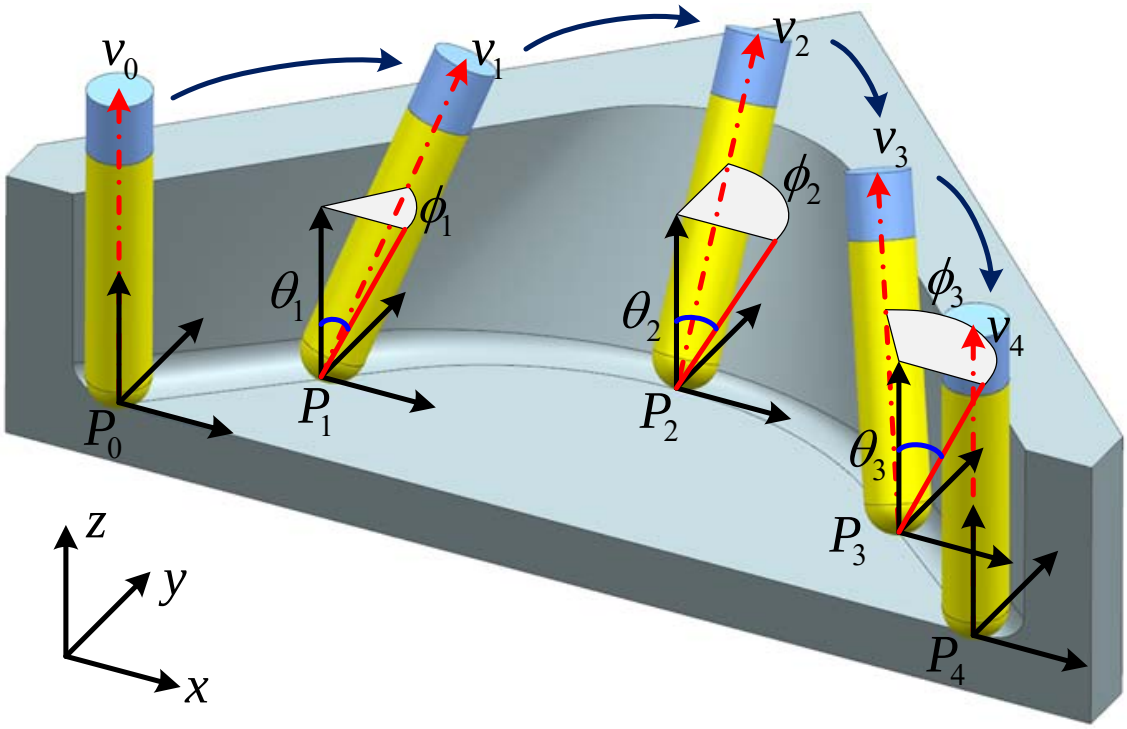


Figure 1.1: Translational and Rotational Degrees of Freedom

of the tool position and orientation is known as the tool pose. A machine tool that is able to completely control the pose of the cutting tool with three translational degrees of freedom (DoF) and two rotational DoF is known as a 5-axis machine tool. 5-axis machines afford engineers with a significantly higher capability to create parts with complex surfaces, but programming and operating these machine tools is extremely complicated and requires a highly trained, experienced user.

1.1 Levels of Control in the Manufacturing Enterprise

The CNC system is an integral part of a larger process planning and execution chain in the manufacturing enterprise, which is defined as a portion of the International Society of Automation (ISA)-95 standard. This standard defines the organizational, operational, and process control subsystems and interconnections of an automated manufacturing process [2]. The CNC system connects the process control level to the manufacturing process it-

self (i.e., it is a bridge from the *cyber* world to the *physical* world) and is responsible for the physical control of the machining process. CNC systems are monitored by supervisory control and data acquisition (SCADA) systems. Operational control is performed by a manufacturing execution system (MES), which is responsible for routing and ensuring the successful completion of orders through the factory. Toolpath generation for the CNC system is performed by the computer-aided manufacturing (CAM) system, which resides in the operational control level. The business level houses the enterprise resource planning (ERP) system, in addition to computer-aided design (CAD) and product lifecycle management (PLM) systems. This control hierarchy is illustrated in Figure 1.2.

1.2 Computer-Aided Manufacturing

When planning a complex machining process, especially for a 5-axis machine tool, users generally employ computer-aided manufacturing (CAM) software to create suitable toolpaths for machining a given part. CAM software enables a user to import the geometry of a part, define the tooling and workholding setup of a certain machining operation, and plan toolpaths to machine the features on the part. Depending on the type of CAM software used, these toolpaths can be as simple as drilling or as complex as multiaxis milling with many coordinated cutting tools [3]. The part models employed in typical CAM software for machining are analytical, meaning that they can be scaled infinitely without a loss of fidelity; models of this form consist of constructive solid geometry (CSG) and boundary representation (B-rep) models that describe a parts geometry as functions of unitless parameter values [4]. For the representation of complex surfaces, these models make use of non-uniform rational basis spline (NURBS) curves that are defined by a sequence of knot vectors and blending functions [5]. The use of NURBS surfaces enables variable-resolution Cartesian progression of a point along the surface by simply varying the precision of the unitless parameters used to interpolate the surface. While this concept would theoretically enable the machining of a surface with infinite precision, this is clearly impossible in prac-

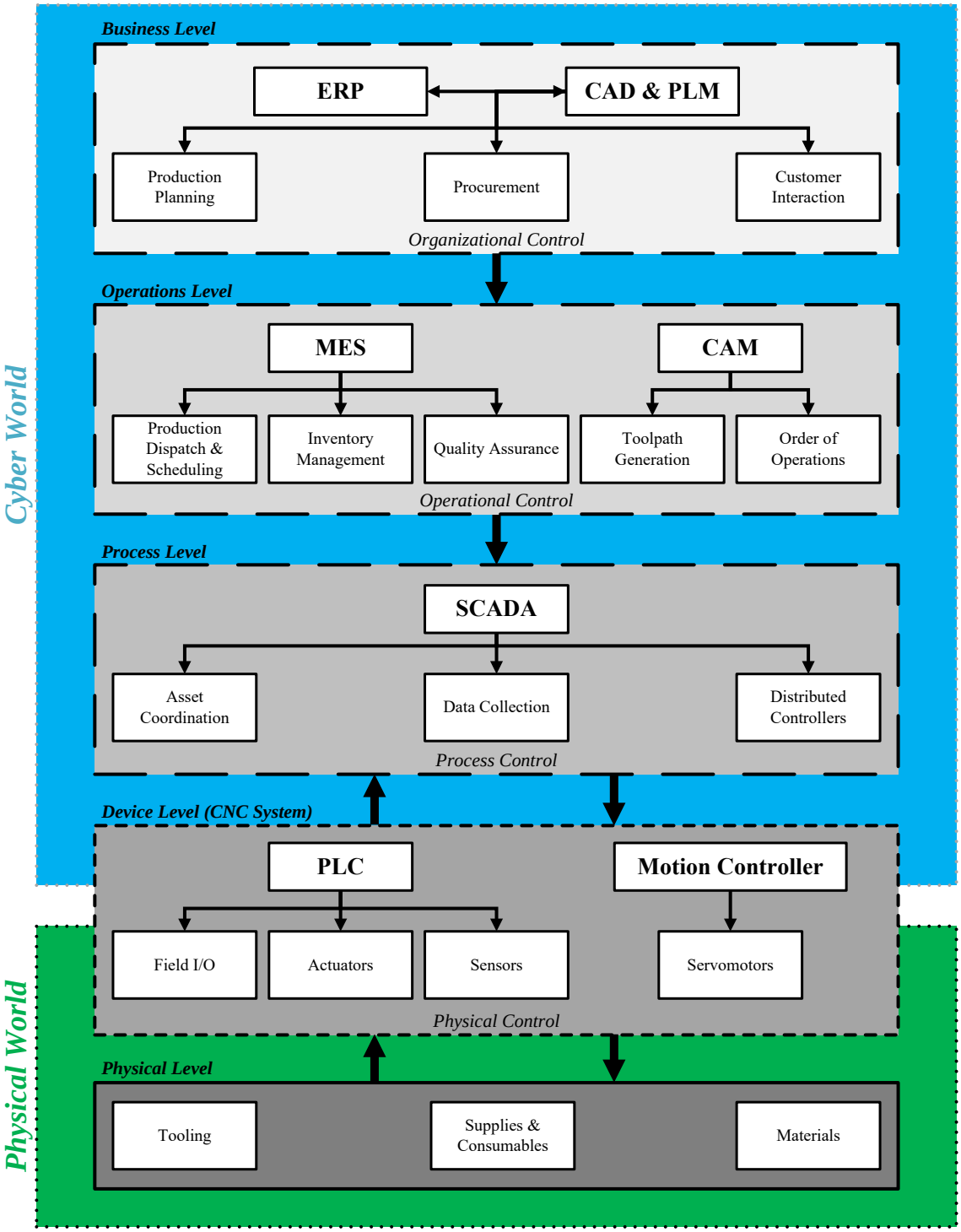


Figure 1.2: Levels of Control in the Manufacturing Enterprise

tice because a CNC machine tool is a complex dynamic system with a digital controller. As a result, discretization of the toolpath is inevitable, leading to physical errors in reproduction of the surface during machining [6].

1.2.1 Voxel-Based CAM

Analytical part models are advantageous in several ways; namely, they can be scaled without losing fidelity and they require small amounts of memory to store. However, analytical models are not ideal when the absolute accuracy of fine surface details is paramount; this is because the complexity of an analytical model is limited by the precision of the computer used to render and operate on the model. This is particularly consequential to the simulation of a cutting process. Take, for example, the representation of scallops on a part surface that would be introduced after a milling operation. To represent each individual scallop with an analytical model would require an extremely complex NURBS formulation (or collection of NURBS formulations) to accurately describe the surface [7]; a better approach is to represent the part surface discretely with many small volumes. This idea is similar to that employed in digital photography: a complex image can be described digitally in terms of picture elements (pixels). If the size of the pixels is small enough, the digital image can recreate its equivalent analog (film) counterpart with sufficient fidelity. In the case of three dimensions, pixels can be extended to voxels. Voxels are cubes whose resolution can be controlled to provide sufficient fidelity in part surface representation. In a typical machining process, the side length of a voxel is on the order of tens of microns. Voxel-based machining has been proven to allow for more complete simulation of the machining process with a reduction of error [8, 9]. This research employs a voxel-based CAM software, known as SculptPrint, that can create toolpaths for 5-axis CNC machine tools [10, 11, 12, 13]. An example surface representation using voxels is shown in Figure 1.3 [14].

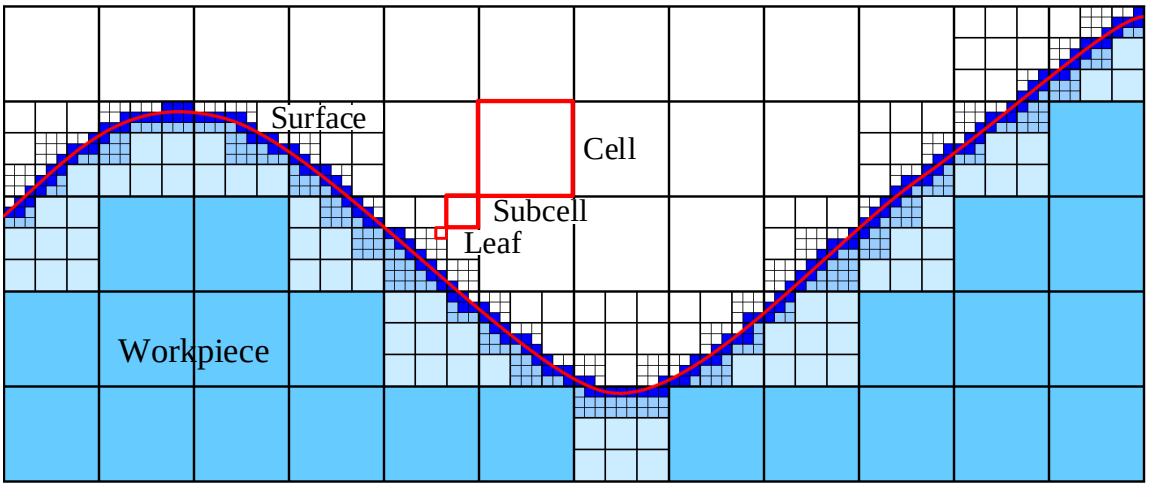
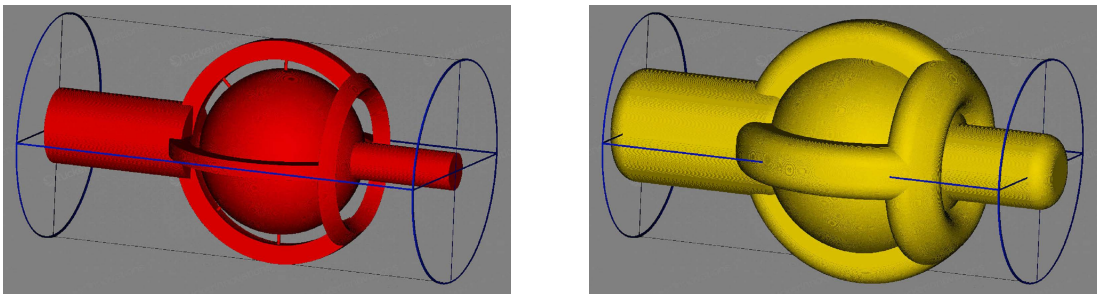


Figure 1.3: Surface Representation with Voxels

1.2.2 Contact Point Creation From Voxel Models

In SculptPrint, the generation of a toolpath begins with the creation of what is known as a contact volume. The contact volume represents the surface along which a given ball-end milling tool can reside without hitting the target part or violating a maximum cutting depth. As a simple example, consider the volumes shown in Figure 1.4 [1]. The target part volume, shown in Figure 1.4(a), is the desired result of a machining operation. Offsetting the part volume by the combination of a machining allowance and the radius of a chosen ball endmill results in the offset volume shown in Figure 1.4(b). The surface of this volume represents the collection of points along which the center of the sphere at the end of a ball endmill can reside without removing material from the target part volume. This example clearly neglects any stock material that remains outside of the target part, as the part volume is offset directly. When accounting for a starting volume that is larger than the target part, the generation of contact volume is performed in the following manner: first, the part volume to be machined is offset in the positive direction (expanded) by the combination of a cutting allowance and the ball radius r of a chosen tool; next, the starting volume is offset in the negative direction (shrunk) by chosen maximum axial cutting depth d; the resulting volumes from these operations are then unioned together to give a surface along which the



(a) Part Volume (b) Contact Volume

Figure 1.4: Part and Contact Volumes for Ball-in-Socket Assembly

chosen ball endmill can reside without either cutting more deeply than the chosen axial depth or removing material from the target volume [15]. A similar algorithm known as 3D contour offsetting is used to create isoscallop toolpaths using the contact volume [16]; an isoscallop toolpath provides a constant cutting width during machining, which allows for higher surface quality than would be achievable with isoparametric toolpath generation approaches [17].

1.2.3 Tool Accessibility Determination

Once the contact volume for a certain part has been created, the orientation of the cutting tool at each point along the surface of the volume must be determined to ensure that the tool does not gouge the part surface. The selection of tool orientations for a multi-axis machining operation is one the most challenging parts of the planning process. Tool orientation selection strives for minimized machining time, and more importantly, collision-free toolpaths and smooth orientation variation [18]. This work employs what are known as accessibility maps, which are a sequence of binary images at every step along a toolpath that define accessible and inaccessible space. The maps are created by considering the complete geometry of the cutting tool and associated hardware, such as a collet and tool holder; this assembly is then checked for collisions with the voxel model at every tool orientation that the machine can produce. Inaccessible space is defined as the group of tool orientations that cause a collision with any piece of the voxel model. With a sufficiently high resolution

of both the simulated part and the maps themselves, collisions can be avoided.

The accessibility maps are governed by the rotational degrees of freedom (DoF) of the cutting tool, which are selectable at every Cartesian point along a toolpath. For a cutting tool with two rotational DoF, a two-dimensional accessibility map can be generated using those DoF as orthogonal dimensions on the map. For example, consider the ball-in-socket assembly and accompanying accessibility maps shown in Figure 1.5. The two rotational DoF, which in this case are referred to as θ and ϕ, each provide a quantized number of orientations for that cutting tool that is dependent on the resolution of the accessibility maps to be generated. The combinations of these orientations can be plotted in two-dimensional space, and each combination of θ and ϕ can be checked for collisions or gouging with the part using a model of the cutting tool assembly. Combinations that result in a collision are shaded, while those that are collision free are unshaded; the resulting binary bitmap then provides the θ and ϕ combinations that the tool is allowed to occupy [12]. By generating an access map for each point along a toolpath and stacking the maps together, a suitable tool orientation progression can be generated. This progression, shown as the red line through the stack of maps, is known as the access path. To ensure smooth variation of tool orientation through the path, the change in angle for each DoF between successive maps is limited. The maps can be generated with variable resolution depending on the curvature of the part to be machined. As the resolution of both the maps and the voxel representation increases, the accuracy and effective usability of the accessibility map algorithm improves.

1.2.4 Toolpath Generation from Voxel Models

Once an isoscallop toolpath has been created and the appropriate tool attitude vectors at each step have been determined, the toolpath must be translated into a form that is readable by a CNC machine tool. Because a voxel model is spatially discrete by nature, it is reasonable to create spatially discretized toolpaths that consist of small linear segments connecting adjacent voxel centers along the toolpath. Because the part surface is three

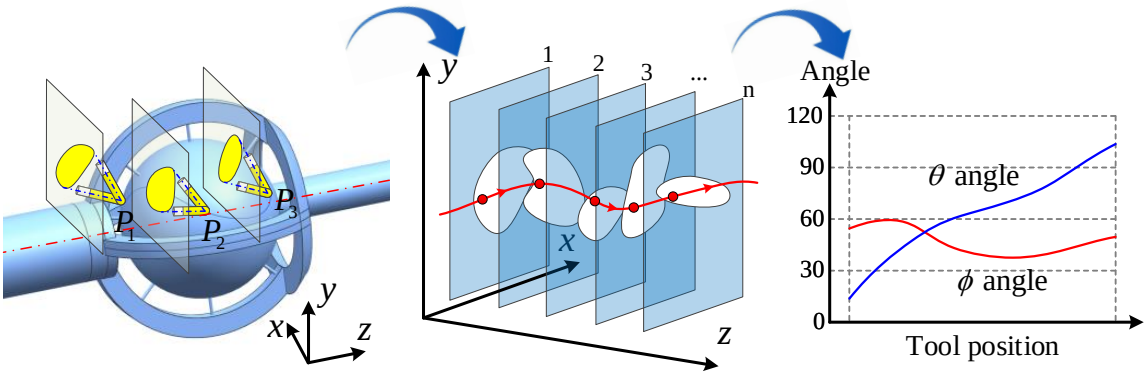


Figure 1.5: Determination of Tool Orientation Using Accessibility Maps

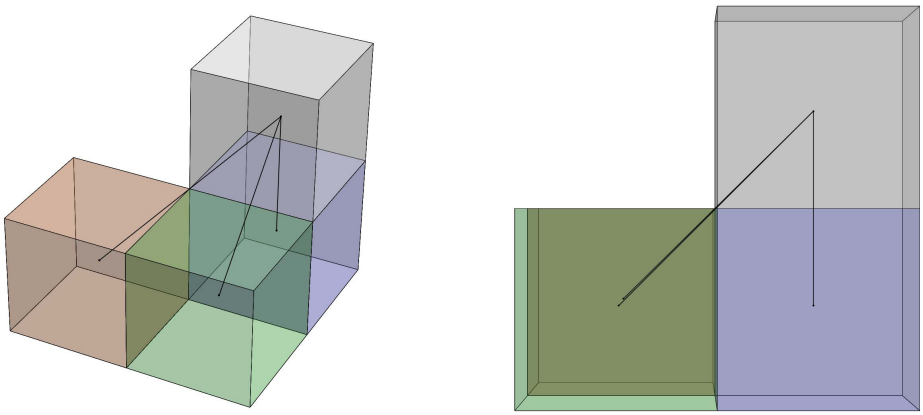
dimensional, there exist three possible segment lengths between voxel centers: the voxel size itself, which is equal to the center-to-center distance of two voxels that share a face; the diagonal distance between two voxels that share only one edge; and the diagonal distance between two voxels that share only one vertex. Therefore, the three possible segment lengths are

$$L_{\text{Shared Face}} = s \tag{1.1a}$$

$$L_{\text{Shared Edge}} = \sqrt{2}s \tag{1.1b}$$

$$L_{\text{Shared Vertex}} = \sqrt{3}s \tag{1.1c}$$

where s is the side length of a voxel and L is the length of the movement segment. These three possibilities for segment length are shown as black lines in Figure 1.6. The first possibility, in Equation 1.1a, is shown as the distance from the center of the gray voxel to the center of the blue voxel; the second possibility, in Equation 1.1b, is shown as the distance from the center of the gray voxel to the center of the green voxel; and the third possibility, in Equation 1.1a, is shown as the distance from the center of the gray voxel to the center of the orange voxel. A toolpath created from a voxel model will consist of many of these movements, which are referred to as steps. Depending on the size of the voxels used for the part model, the steps can be very short and the entire toolpath can consist of



(a) Isometric View (b) Front View

Figure 1.6: Three Possibilities for Segment Length

millions of individual steps.

Discrete geometric representations cannot be scaled up without loss of fidelity; in other words, voxel models have finite resolution. An analytical geometric representation, on the other hand, can be scaled infinitely (disregarding the fact that a computer cannot process an infinitely large number) and no loss of fidelity will be observed. However, because the control system for a CNC machine tool is digital, the analytical model will ultimately need to be discretized, either spatially or temporally, for it to be processed on the machine. As along as the voxel size is as small as or smaller than the resolution of the CNC machine tool used to recreate the surface, and disregarding any complicating factors in reading many small linear movement commands to define the surface, the machined result will be the same as it would be with an analytical model. Therefore, the use of a voxel model of sufficient resolution will not produce an inferior physical recreation than an analytical model.

1.2.5 Material Removal Rate (MRR) Analysis with Voxelized Toolpaths

Simulation and analysis of the physics of the cutting process is highly computationally intensive when applied to 5-axis operations [19]. A typical analysis to perform when evaluating the viability of a given toolpath is computation of the material removal rate (MRR)

profile along the path; this analysis gives an indication of not only the efficiency of the toolpath in removing material from the workpiece, but also whether or not the selected cutting tool is capable of machining a given toolpath with the specified parameters. MRR analysis is computationally intensive when performed on analytical part models, but quite simple when done using voxel models [9].

MRR analysis within voxel-based CAM is performed as follows. Once the appropriate axis commands are determined for a given step, a cutter model is swept along the step to determine the amount of material removal over the step. The total volume removed at a certain step i is simply the summation of the number of voxels that touch or reside within the cutter envelope C at the end of the step,

$$V_i = \sum_i dv \times s^3 \,|\, dv \in C \tag{1.2}$$

where V_i is the total volume that is removed during the step i, s is the side length of a voxel, and dv is a voxel which is contained within the set of all Cartesian points that define the cutter envelope C. As the cutter volume is swept along the path, its pose changes at each step according to the movement direction of the path and the tool orientations that were assigned during accessibility analysis. The instantaneous material removal rate from the voxel model can then be calculated by

$$MRR_{\text{Instantaneous}} = \frac{V_i}{\Delta t} \tag{1.3}$$

where Δt is some time interval. By enforcing the constraint that the instantaneous MRR should always be equal to some upper bound, Equation 1.3 can be rearranged to find the time interval over which the MRR is computed. Exact determination of the MRR limit of a certain milling cutter is beyond the scope of this dissertation, as it requires extensive force analysis that is a function of cutter geometry, cutting parameters, machine tool compliance, and workpiece material [20]. However, previous work has demonstrated that constraining

the MRR of a certain tool to a threshold determined by tool manufacturer recommendations is a valid method to avoid tool breakage [21, 22]. This method uses the maximum values of feedrate and cutting depth as provided by the tool manufacturer for a given material to compute the MRR limit for a square nose endmill according to

$$MRR_{\text{Limit}} = V_f a_p a_e \tag{1.4}$$

where V_f is the feedrate, a_p is the axial cutting depth, and a_e is the tool engagement, or radial cutting depth. The feedrate and cutting depth parameters can be looked up in a table or are provided by the tool manufacturer. Equation 4 can also be formulated in terms of chip load per tooth and spindle speed, but for the purposes of controlling the tool velocity independently of the spindle, it is more useful in its present form. However, this technique is only valid in the presence of some assumptions: first, the tool must be experiencing an end milling condition in which the majority of cutting force is exerted at the tip of the tool; and second, the cutting parameters that are given are the maximum allowable values that will never result in tool breakage [21]. For the purposes of endmilling with a ballend tool, as is done in this dissertation, these assumptions are reasonable. An example of MRR calculation using a voxelized part model is shown in Figure 1.7. A ball endmill, shown in light blue, is tracing the toolpath on the front of a ball-in-socket assembly. The plot in the bottom left of the Figure shows the volume that is removed at each step of the toolpath; the X-axis of the plot denotes the step number, while the Y-axis denotes the total volume removed for that step. It is apparent that, if this curve were flattened such that the volume removed at each step was equal to the MRR limit of the ball endmill, the peaks and valleys in the MRR curve could be removed.

One issue arises in enforcing the constraint that the MRR always be equal to the upper limit of MRR for a given cutting tool; namely, the kinematic limits of the machine tool may not be sufficient to reach a certain Δt. For example, if the solution to Equation 1.3 when the

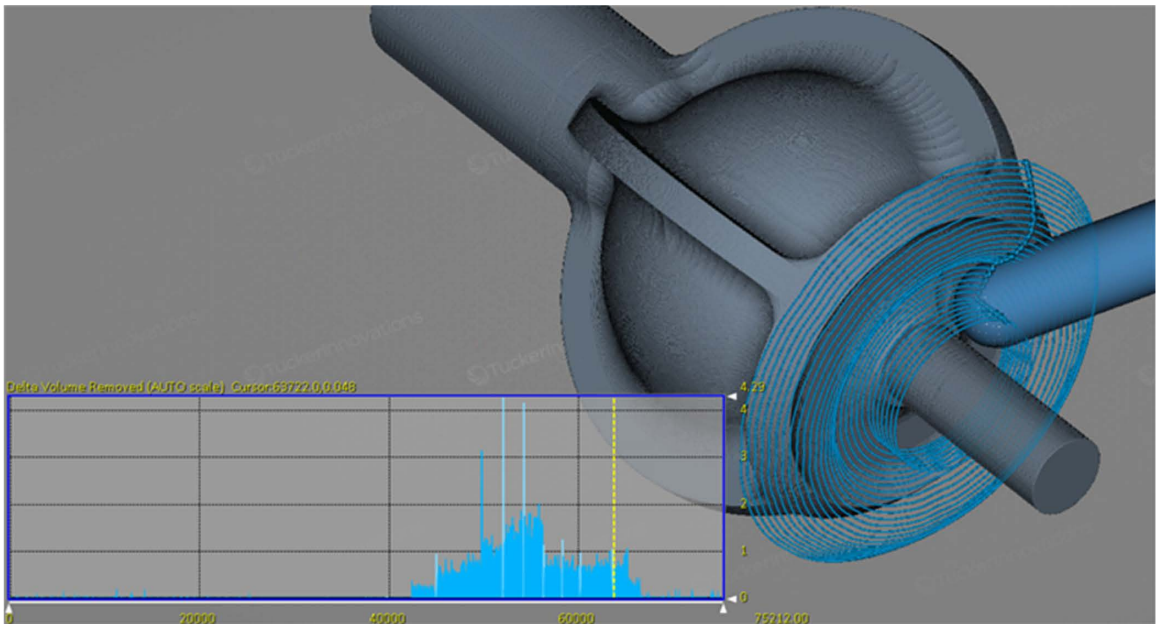


Figure 1.7: MRR Analysis of a Toolpath Derived from a Voxel Model

instantaneous MRR is set equal to the MRR limit results in a sufficiently small Δt over the step length, the resulting velocity required to achieve the move in the given Δt may exceed the maximum velocity of one or more axes of the machine. Therefore, the time increment must be limited according to the axis limits of the machine tool. The kinematic constraints of the machine are discussed in later sections.

1.2.6 Virtualization of GPU-Accelerated CAM

The requirement that the workstation of the SculptPrint user be equipped with a powerful GPU can be a limitation in some cases, and can decrease the accessibility of the software to a wide range of users. A virtualized testbed has been implemented to evaluate the feasibility of providing voxel-based CAM access form a cloud platform. This system was developed to explore Desktop-as-a-Service (DaaS) as a potential solution to increase the agility of a manufacturing operation. As manufacturing industries move towards a service-oriented approach, where highly dynamic production facilities must be able to respond quickly to rapidly changing market conditions, the traditional model of data management on the shop

floor has been shown to be slow to respond and lacking in efficiency [23, 24, 25]. The current trend of moving computing to cloud resources, which includes cloud computing (CC), Infrastructure-as-a-Service (IaaS), and DaaS, has been extended to the manufacturing realm as cloud manufacturing (CM). CM provides the framework for compute and data management resources necessary for manufacturing to be moved to a cloud platform [26]. These resources can include product data management (PDM), ERP, process monitoring, and computer-aided manufacturing (CAM). CM enables a manufacturer to focus time and investment into capital that is directly responsible for producing goods, rather than auxiliary hardware such as servers and databases. The virtualized testbed is an extension of CM into the realm of voxel-based CAM.

Previous work has demonstrated that the DaaS system is capable of providing high-performance SculptPrint access to any Internet-connected device [14, 27]. This system allows multiple simultaneous users to share a single server-class GPU (NVIDIA GRID K1) for toolpath planning and simulation. This system was implemented using Citrix XenServer which provides multiple Windows Server 2012 virtual machines (VMs) to users. Recent improvements to the system have been performed to equip it with an NVIDIA Tesla M60 GPU, which exhibits far superior performance than was possible with the K1. Each virtual machine is equipped with a dedicated GPU, two virtual CPUs, and 64GB of RAM. The use of Windows Server 2012 as the VM operating system (OS) enables hosting of multiple simultaneous user sessions on each VM, where sharing of the VMs GPU is managed by the OS and the GPU driver. This implementation has enabled the use of SculptPrint in an educational setting, where students used the software for toolpath planning and design for manufacturability (DFM) training on their personal laptops and tablets [28, 29, 30, 31, 32]. This system can be used not only for toolpath planning, but also for thorough analysis of commanded and actually-executed toolpaths. Specifically, MRR can be computed to voxel resolution over the course of a toolpath, and the actually instantaneous MRR that was achieved while executing the path can be computed by implementing additional

functionality to read data collected during machining into SculptPrint.

1.3 Machine Tool Programming

Description of a toolpath to a machine tool controller is most frequently accomplished through the industry standard RS-274D (ISO 6893), also known as G-Code [33, 34]. In its most basic form, G-Code consists of commands that instruct a machine tool to move its axes to realize a given tool motion. For example, a simple linear movement can be accomplished by providing the endpoints of the move and a prescribed velocity. Other commands in the RS-274 standard include those for performing typical auxiliary operations necessary in the machining process, such as turning the coolant system on or off. Many of these commands are referred to as M-codes. A complete G-Code program for making a particular part is composed of many lines of G- and M-codes which describe the exact steps the machine tool must follow (including tool changes, toolpath geometry, etc) to machine the part. For machine tools with only translational axes, such as typical machining centers, the creation of G-Code movements to describe the geometry of a toolpath is relatively simple; the G-Code program only needs to move through points in 3-dimensional space. However, for machine tools with multiple translational and rotary axes, such as 5-axis machine tools, the creation of G-Code to describe a toolpath becomes prohibitively complicated to perform manually due to the large number of variables involved. For example, not only does the pose of the cutting tool need to be specified at every point along the toolpath, but also the toolpath must be checked for collisions with either the workpiece or the machine tool structure [35]. A CNC machine tool interprets G-Code in a serial fashion, where lines containing movement commands, known as a blocks, are executed in the order they are commanded. A typical block of G-Code for a 5-axis move, denoted here as N_i, is

$$\mathrm{N}i\text{: G1 X}x\ \mathrm{Y}y\ \mathrm{Z}z\ \mathrm{A}a\ \mathrm{B}b\ \mathrm{F}V_f$$

where x, y, z, a, and b are the end points of the move for each respective axis and G1 specifies that all axes should be linearly interpolated from the start points to the end points

in such a way that all axes complete the move at the same time. In the present case, the end points of the move denote absolute axis positions, which do not necessarily correspond to the pose of the tool with respect to the workpiece coordinate system. The F command is used to specify the feedrate V_f, which is interpreted in different ways depending on the currently active modal commands of the program. There are three possible scenarios in which the F command can be interpreted:

1. If the machine tool is currently in synchronous feed mode (G95), where the movement speed of the axes is synchronized with the spindle rotation speed, the F command denotes the Cartesian movement distance along the block per spindle revolution. Note that synchronous feed is frequently unsupported for simultaneous 5-axis movements.

2. If the machine is in feed per time mode (G94), the F command denotes the maximum allowable Cartesian movement velocity during the move. Whether or not the tool velocity actually reaches this maximum limit is dependent upon the trajectory planner, which will be discussed in later sections.

3. If the machine is in time or inverse time feed mode (G93), the F command denotes either the amount of time allotted to complete the move or the inverse of that time value, respectively. This method is preferred for 5-axis machining, as it allows the translational and rotational axes to be treated similarly [36].

Regardless of which modal feed command is active (G95, G94, or G93), the actual realized tool velocity is dependent on a number of factors, and may not necessarily correspond with the programed feedrate.

1.3.1 Tool Space and Joint Space

The previous discussion was limited to toolpaths in which the coordinates are described to the machine in what is known as joint space, which means that the given movement

commands correspond directly to axis positions of the machine tool. Because toolpaths specified in joint space command the physical axes of the machine tool directly, they are dependent on both the structure of the machine and the fixture location of the workpiece with respect to the table center of rotation. Many modern CNC control systems can also interpret toolpaths in what is known as tool space or workpiece coordinates (WPC), in which the pose of the cutting tool with respect to the workpiece origin is given instead of the absolute axis positions. The machine tool controller then translates this tool space point in to a joint space point using the inverse kinematic transformation (IKT) for the structure of the machine tool. Many machine tool builders refer to this mode as tool center point control (TCPC) mode [37]. For the particular 5-axis machine tool considered in this disseration, which has two rotational axes on the table (known as a table-table machine), the IKT can be expressed as [38]

$$\begin{bmatrix} X_j \\ Y_j \\ Z_j \end{bmatrix} = \begin{bmatrix} \sin B & -\sin A\cos B & \cos A\cos B \\ \cos B & \sin A\sin B & -\sin B\cos A \\ 0 & \cos A & \sin A \end{bmatrix}^{-1} \begin{bmatrix} -X_t - 4.0697\cos A\cos B + 1.0023 \\ -Y_t + 4.0697\sin B\cos A \\ -Z_t - 4.0697\sin A - 0.372 \end{bmatrix} \quad (1.5)$$

where the t subscript denotes a point in tool space and the j subscript denotes a point in joint space. This particular formulation is for a machine tool known as the PocketNC, where the A axis rotates about an imaginary axis that is parallel to the X axis and the B axis rotates about an imaginary axis that is parallel to the Y axis. This machine tool will be described in later sections. In order to determine position of the tip of the cutting tool in the workpiece frame of reference when given joint positions (for feedback purposes, for example), the forward kinematic transformation (FKT) is used. The FKT is simply a

rearrangement of Equation 1.5,

$$\begin{bmatrix} X_t \\ Y_t \\ Z_t \end{bmatrix} = \begin{bmatrix} -4.0697\cos A\cos B + 1.0023 \\ 4.0697\sin B\cos A \\ -4.0697\sin A - 0.372 \end{bmatrix} - \begin{bmatrix} \sin B & -\sin A\cos B & \cos A\cos B \\ \cos B & \sin A\sin B & -\sin B\cos A \\ 0 & \cos A & \sin A \end{bmatrix} \begin{bmatrix} X_j \\ Y_j \\ Z_j \end{bmatrix} \quad (1.6)$$

The use of TCPC allows a CAM system to specify toolpath positions in the coordinate system of the workpiece, which allows the resulting G-Code to be independent of the machine tool structure and the absolute location of the workpiece origin with respect to the table center of rotation. While this is convenient for low-volume work that is not optimized for production time, the CNC unit is forced to solve the IKT at every toolpath point and make all decisions about completing the move; this may slow down the axes to account for the additional computation time, causing the actual feedrate to deviate from the programmed feedrate. As a result, the use of TCPC is limited to situations where the convenience of isolating the G-Code from the machine structure and the absolute position of the workpiece is deemed to be more important than absolute production throughput [39].

1.3.2 Kinematic Constraints

Regardless of how the toolpath points are described to the machine, each move is subject to a set of constraints that depend on both the kinematic limits of the machine and the cycle time of the CNC unit. These constraints, on a per-axis basis for the block N_i, can be summarized as follows [40]:

$$|v_i| \leq \min(V_f, V_{\text{Cycle}}, V_{\text{Max}}) \quad (1.7a)$$

$$|a_i| \leq A_{\text{Max}} \quad (1.7b)$$

$$|j_i| \leq J_{\text{Max}} \quad (1.7c)$$

where v_i, a_i, and j_i are the actual velocity, acceleration and jerk for the block N_i, respectively; V_f is the commanded feedrate for the block N_i; V_{Cycle} is the maximum tool velocity allowed by the cycle time of the CNC; and V_{Max}, A_{Max} and J_{Max} are the maximum possible velocity, acceleration and jerk for the axis under consideration, respectively. V_{Cycle} is simply equal to the distance of the move divided by the CNC cycle time. As a result of these constraints, the machine may not reach the programmed feedrate during the block N_i. Additionally, the kinematic limits of the slowest axis limit the realizable velocity over the entire move, as all axes must arrive at the end point at the same time.

1.3.3 Advanced G-Code Functionality

Modern machine tool control systems have the capability to perform more sophisticated operations programmatically, such as loops and conditional statements, in addition to the simple movement commands described above [41, 42, 43]. Additionally, the interpolation of various types of splines is possible using specialized and often controller-specific G-Codes. For example, the Mitsubishi M800W control system that forms the basis of Mazak's Smooth control architecture provides a G-Code for NURBS interpolation given knot vectors and control points for the spline [42]. In order to successfully utilize this feature, the CAM system used to generate a toolpath must be capable of outputting NURBS commands, or the programmer must be proficient enough to calculate and program the splines manually. Regardless of the level of sophistication of the CAM software, the generated toolpath still must be translated to a format that is readable by the machine tools control system; almost always, this format is G-Code.

NC programming is now considered the bottleneck of realizing high performance machine tools for a number of reasons [44]. First, translation of a toolpath into G-Code creates a one-way link between the CAM system and the machine tool that does not allow for bidirectional data flow, and second, G-Code does not provide the CAM system with complete control over all motion of the machine tool [45]. When a CAM system creates a toolpath,

it has far more knowledge over the physics of the path (MRR, etc.) than it is able to communicate to the machine tool through G-Code [46]. G-Code can only communicate the maximum feedrate permissible during a block (or time to complete the block), and thus cannot specify a target velocity profile to be followed. The machine tool will interpret the block of G-Code and decide on a velocity profile to apply to that block. Usually, the velocity profiles that are followed by the machine are of the trapezoidal type, meaning that the axis will accelerate with constant magnitude up to some velocity, cruise at that velocity for some period of time, and then decelerate to finish at the end of the block. If the next block commands a turn or change in velocity, the machine must compute the necessary trajectory in real time to realize the programmed motion without violating the kinematic constraints from Equation 7. The controller is forced to compute velocity profiles for each block which does not allow the CAM system to have complete control over the positional derivatives of each axis. Depending on the velocity profiles that the controller assigns to each axis, high impulses on the tool that decrease the surface quality can be experienced [47]. Additionally, the lack of process feedback to the CAM system from the machine tool results in the loss of valuable information that could be used to improve the toolpath, such as spindle power consumption, actual axis velocities, and program execution time. An alternative toolpath representation is needed to allow for high-density bidirectional data flow between the CAM system and the CNC itself. This representation should not only allow the CAM system to have complete control over all axis motions, but also it should provide a framework for feeding back process data that can enable toolpath optimization in the CAM system.

1.3.4 STEP-NC

To address some of the common limitations and criticisms of typical G-Code programming of machine tools, researchers have developed a new NC programming strategy known as STEP-NC or AP238 [48]. STEP-NC extends the well-known STEP standard for CAD

data interchange to include machining process instructions that are defined in ISO 14649 [49]. STEP-NC is an object-oriented machine tool programming standard that defines a machining process by individual features rather than machine movement [50]. As a result, STEP-NC enables interoperability between disparate machine tools: each machine can use the same STEP-NC program and simply decide how to perform each operation based on the configuration of the machine interpreting the program [51]. STEP-NC was developed to address the major shortcoming of RS-274D that in-process data cannot be relayed from the machine tool to a CAM system; because the STEP-NC file can be easily exchanged through multiple pieces of equipment on the shop floor, it can be modified by one user or machine if a problem with the program is detected and the resulting new STEP-NC file can be immediately executed on another machine on the shop floor [52]. However, the use of STEP-NC requires a sophisticated interpreter on the machine tool that is able to generate low-level movement commands for the machines axes from the object-oriented STEP-NC file [53].

While the use of STEP-NC can alleviate some difficulties in data interchange between CAM and CNC, it still does not allow the CAM system complete control over the motion of the machine tool; rather, complex surfaces are represented parametrically in a STEP-NC file and the CNC system still must plan trajectories in real-time to machine those surfaces [54]. In some cases, the surfaces are still described in the STEP-NC file as points as they would be with traditional G-Code, but they are simply wrapped into the STEP-NC format [55]. An alternative XML-based representation known as Numerical Control Markup Language (NCML) has also been demonstrated; NCML is structured similarly to STEP-NC, although it allows easier exchange of machining plans over the Internet [56, 57]. As is the case with STEP-NC, low-level movement commands are still left up to the controller to determine, and thus the CAM system does not have direct control over all tool motion.

1.4 Process Data Feedback from Machine Tools

Due to the proprietary nature of the vast majority of CNC systems, access to in-process data from the machine is often limited. The recent trend towards interoperability of shop floor equipment driven by increasing adoption of Industry 4.0 concepts has led to the development of standards and protocols for collecting additional data from machine tools [58].

1.4.1 MTConnect

MTConnect is an open and royalty-free eXtensible Markup Language (XML)-based standard that defines the format and naming of data transmitted from a piece of manufacturing equipment. MTConnect is developed by the MTConnect Institute and has found acceptance among both manufacturers and control builders alike [59]. MTConnect follows a REpresentational State Transfer (REST) over HyperText Transfer Protocol (HTTP) interface where a client will initiate a request for data from an MTConnect-enabled device and then wait for the requested data to be returned. The MTConnect-enabled device exposes available data through a piece of software known as an MTConnect agent, which in turn logs data coming off a machine tool through an MTConnect adapter. On modern machine tools, the adapter is a piece of software that interfaces with the machine's control system; for legacy machine tools, discrete hardware adapters can be used if the control system does not support a software adapter. The MTConnect standard also enables streaming of realtime data from the agent, which eliminates the need for the client to poll the agent for updated data [60]. Regardless of whether polling or streaming is chosen, however, MTConnect is strictly a read-only protocol and supports only data collection, not machine command transmission. Although the read-only nature of MTConnect is by design, implementers of MTConnect-enabled manufacturing systems must maintain two separate pathways for data transmission: the forward pathway carries machine commands (e.g., in the form of G-

Code), and the feedback path carries process data in the MTConnect format. An example architecture of an MTConnect-based monitoring system with a PC-based CNC is shown in Figure 1.8.

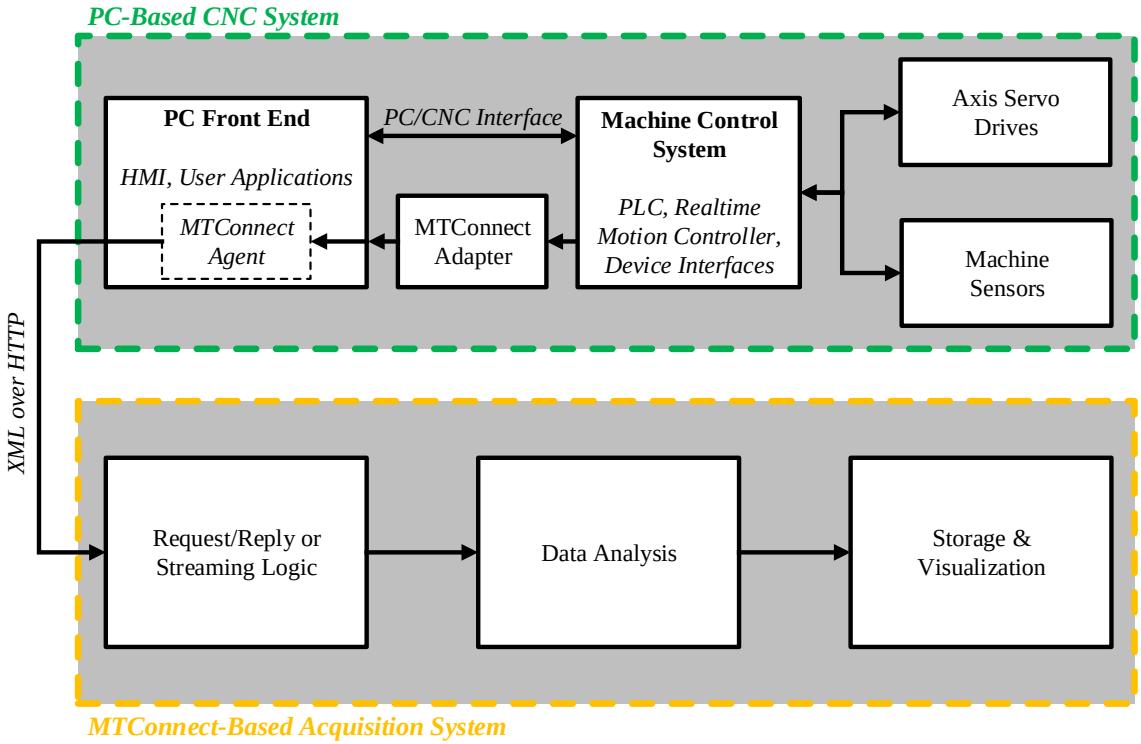


Figure 1.8: Example of a Typical MTConnect System Architecture

Both research and industrial communities have demonstrated significant interest in deploying MTConnect as a means to collect process data from manufacturing equipment. A large body of work is focused on on collection of data using a discrete data acquisition system for the purposes of process improvement by either plant personnel or a supervisory control system [61, 62, 63]; other works have studied realtime machining process improvement with MTConnect data [64]; monitoring of additive manufacturing equipment running on open-source controllers [65]; deployment of Internet-of-Things (IoT) devices for the collection and transmission of MTConnect data [66, 67, 68]; use of both widely-adopted and open-source software platforms for data collection [69]; the use of MTConnect to correlate intended product data with actually realized process data [70, 71]; integration

of process and metrology data [72]; and performance and quality-of-service implications in MTConnect deployments [73]. Numerous commercial solutions that leverage MTConnect data for process monitoring and dashboard visualization, such as Memex MERLIN, TechSolve ShopViz, FORCAM Force, and System Insights VIMANA are also in use in production environments [74].

Many machine tool builders have implemented MTConnect compatibility on their CNC systems, but the variety and frequency of data available through MTConnect is still limited [75, 76]. Specifically, most MTConnect implementations can provide update rates of the order of tens of Hertz, which is not sufficient for capturing high density axis data that is necessary for the evaluation of the machines motion control system. To enable more thorough analysis of high density toolpaths, such as those created from voxel models, an alternative data acquisition method is required to capture axis data at the frequency with which the motion control system operates.

1.4.2 OPC-UA

Another communication standard of interest to researchers and developers in industrial automation is known as OPC-UA [77], which enables data exchange between various levels of the process planning and execution chain [78]. In contrast to MTConnect, which enables semantic interoperability, OPC-UA provides syntactic interoperability; as a result, OPC-UA provides a syntax for data exchange, while MTConnect actually provides meaning for data that are being exchanged. OPC-UA, which is maintained by the OPC Foundation (where OPC was originally known as Object Linking and Embedding (OLE) for Process Control, but is now simply Open Platform Communications), is an evolution of the original OPC standard that is based on Microsoft's Distributed Component Object Model (DCOM). OPC-UA was developed to address concerns with the proprietary nature of DCOM and to increase extensibility of the standard to cover additional devices and systems that were not possible to integrate into OPC [79]. OPC-UA adopts a service-oriented architecture (SOA)

and defines a standard data format for the exposure of actions and attributes for a compliant device in a unified data model. Communication of OPC-UA data is accomplished using either XML (known as UA Web Services) or binary (known as UA Native) communication methods between OPC-UA clients and servers. The OPC-UA standard defines only the format for messages that are passed between clients and servers, and does not provide a standardized application programming interface (API) for implementing a complete OPC-UA stack; as a result, it is the responsibility of the systems integrator to develop a suitable API for a given device [80].

Current research directions with OPC-UA have been more varied than those with MTConnect for two primary reasons: firstly, the original OPC standard has been in existence for longer than MTConnect and OPC-UA builds upon the momemtum of OPC; and secondly, the syntactic interoperability provided by OPC-UA enables interconnection of a wide range of devices with user-defined data models [81]. Thus, implementers of OPC-UA do not have to rely on the relatively slow standards development process to add additional data items to the standard (as is the case with MTConnect), and can instead simply define data models as necessary. While the lack of semantic interoperability when using OPC-UA can enable more rapid deployment to a variety of systems, it does not ensure that all devices conforming to the OPC-UA standard can actually communicate effectively. As a result, research in the use of OPC-UA for control and monitoring of an industrial process has ranged from pharmaceutical manufacturing [82] to aluminum rolling [83] to power generation and distribution [84]. In the discrete manufacturing area, research has focused on the development of an architecture to use OPC-UA as a means to enable data exchange between vertically-separated systems in the process planning, control, and execution chain (e.g., ERP, MES, and CNC systems) [85, 86]; development and implementation of data acquisition systems based on IoT platforms that rely on OPC-UA for data transmission [87, 88]; control and monitoring of a flexible manufacturing system for machining and assembly [89]; and predictive model construction based on process data gathered using an OPC-UA

stack [90]. In contrast with MTConnect, OPC-UA and simplified versions of the OPC architecture also enable the transmission of control commands to manufacturing equipment, which has been demonstrated as a means to operate machine tools remotely [91, 92].

1.5 Trajectory Control for CNC Machine Tools

1.5.1 Machine Tool Control Systems

By its very nature, a computer-numerical control system relies on a digital computer to perform the necessary computations required drive the servo axes of the machine along a programmed toolpath. Machine tool control systems are typically proprietary and not easily modifiable, with the exception of the Enhanced Machine Controller (EMC) project that was originally developed by the National Institute of Standards and Technology (NIST) and has since fostered a thriving open-source community that develops the LinuxCNC and Machinekit spinoffs [93]. The vast majority of CNC systems in use today, such as those manufactured by FANUC, Mitsubishi, and Siemens, are closed-source and provide only limited opportunities for modification and functionality extension. Regardless of the level of openness of the CNC, virtually all controllers have similar architecture. A machine tool control system typically consists of three functional elements: a human-machine interface (HMI) that displays the current operating state, a motion control system that is responsible for interpreting and executing the programmed tool motions, and the servo drives that contain the power electronics necessary to control the servomotors on each axis of the machine.

The HMI is typically not a realtime component, and is frequently implemented on a standard PC operating system. The motion control system, however, must operate in realtime as it plans trajectories and executes the toolpath. Realtime operation of the motion control system can be accomplished with the use of a realtime operating system (RTOS), where execution of certain operations can be given priority to complete in a deterministic fashion [94]. Depending on the type of servo drives that are used, the motion controller may

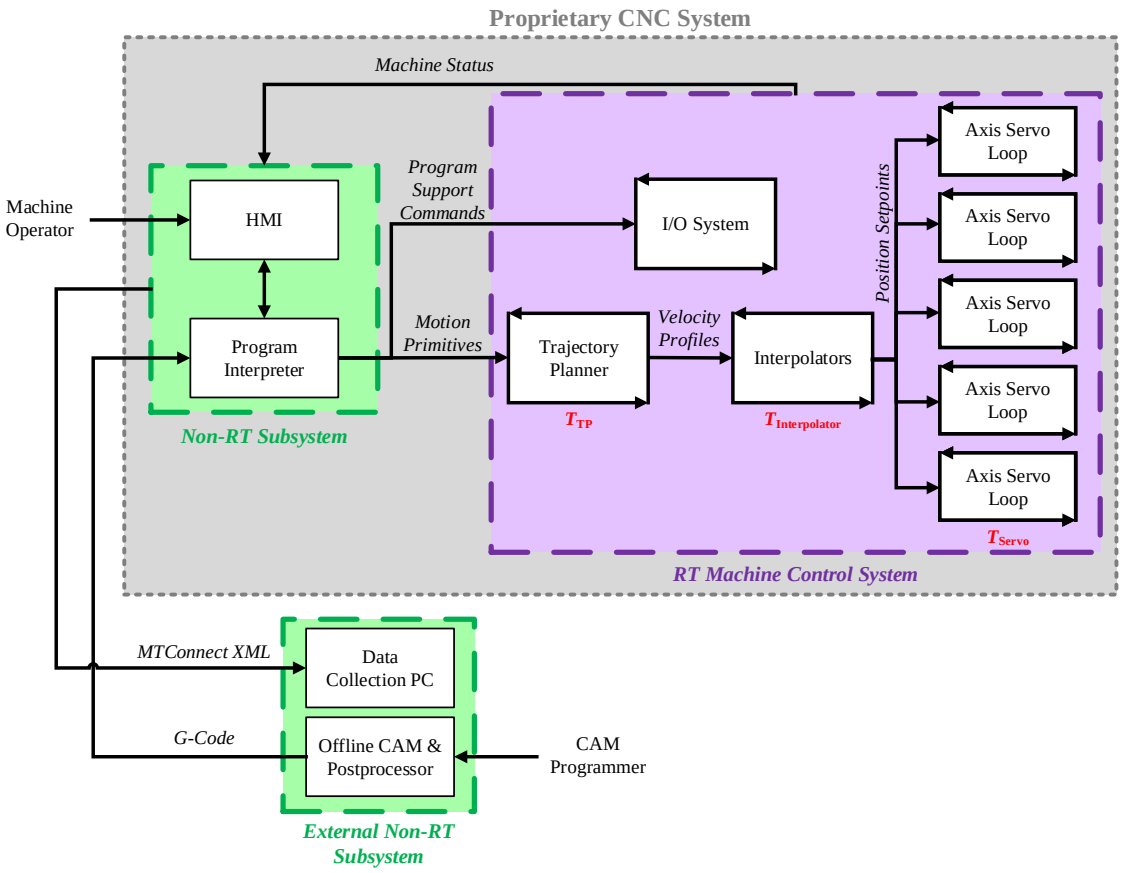


Figure 1.9: Overview of Machine Tool Control System

or may not be responsible for the actual feedback control of the axis servomotors; on newer digital servo drives, the drive itself reads the feedback device of the servomotor (typically a rotary or linear encoder) and uses standard proportional-integral-derivative (PID)-type control of both velocity and position to ensure that the servomotor tracks the commanded path [95]. The primary area of interest for this dissertation is the trajectory control system, which consists of a program interpreter, a trajectory planner, and an interpolator. Data transfer between these elements of the control system is typically accomplished using a shared memory buffer, which allows the components to be run at different rates and on different threads but still share information. Figure 1.9 shows the high-level overview of a typical machine tool control system architecture, each relevant component of which is explained below.

1.5.2 G-Code Interpreter

The first stage in trajectory generation for a CNC machine tool is interpretation of the program commands, which are output from a CAM system as G-Code. These commands can be movement commands, such as the G1 command presented above, or they can be auxiliary commands, such as M-Codes. The interpreter reads successive lines of commands from a file or buffer and converts them to primitives in a motion queue [96, 97]. This approach has the advantage of allowing complex G-Codes, such as canned cycles, to be represented to the machine tool by a single line of code. As a simple example, consider the G2 command which represents clockwise circular interpolation on some preset workplane. If the buffer contains a block with a G2 about some arbitrary center, the interpreter will add this command to the movement queue as a circular interpolation command. Similarly, if a canned cycle is programmed, such as peck drilling, the interpreter will decompose the canned cycle into the correct sequence of linear feed and rapid traverse movements to realize the peck drilling cycle and add each move to the movement queue. Note that the movement queue simply contains a list of primitives as interpreted from the program, all of which can vary in length and programmed velocity. It is the responsibility of the trajectory planner and the interpolator to actually drive the machine axes along the primitives residing in the motion queue.

1.5.3 Trajectory Planner

The next step in toolpath execution is to assign positional derivative profiles to each of the primitives in the motion queue. The trajectory planner (TP) must assign appropriate derivative profiles that are realizable by the machine tool, meaning these profiles must respect the limits given in Equation 1.7. The trajectory planner is also responsible for blending subsequent movement segments together to ensure that the feedrate stays above zero while keeping positional error within some prescribed bound. Additionally, the trajectory planner must react to changes in a speed control input, known as the feedrate override, that is

given by the operator to uniformly scale the maximum programmed velocity (the F command) by some factor. As a result, the trajectory planner is a realtime component of the motion control system that is run online during machining.

In many machine tool controllers, simple trapezoidal velocity profiles are assigned to each movement primitive; these velocity profiles consist of a constant acceleration phase, a cruise phase, and a constant deceleration phase. An example of such a trapezoidal profile is shown in Figure 1.10 for a single axis. The axis accelerates with the maximum amount of acceleration, a_{Max}, to a velocity V_{Cruise}; the axis maintains V_{Cruise} until it begins to decelerate with $-a_{\text{Max}}$ to come to a stop at the end of the move. Two characteristics are notable about this type of velocity profile: first, the integration of the trapezoidal profile results in a quadratic position profile during acceleration and deceleration phases, which causes rounding of sharp corners in multi-axis movements; second, the constant acceleration phases result in jerk impulses to the axis which are not physically realizable due to the electrical time constant caused by the inductance of the motor controlling the axis [98]. This is an example of a bang-bang trajectory planning scheme, where one of the positional derivatives of the axis is at its limit for the duration of the move [99, 100].

Trapezoidal profiles are frequently used because they are easy to compute in realtime. Wilson showed that trapezoidal velocity profiles result in minimum time trajectory planning (MTTP) for controlled motion if the infinite jerk assumption is valid [101]. More recent work by Li, et al. proved that, for jerk-limited trajectories, a bang-bang scheme where the jerk is constrained to the actuator limit during the acceleration phase results in the solution to the MTTP problem [102].

Online planning of trajectories allows a machine to react to unexpected changes in the feedrate override by the operator, but can be limiting when attempting to machine very dense trajectories. To enable high tool velocity along the path while simultaneously providing accurate positional following capability, modern machine tool control systems employ a lookahead buffer that reads a number of blocks ahead of the current machine position

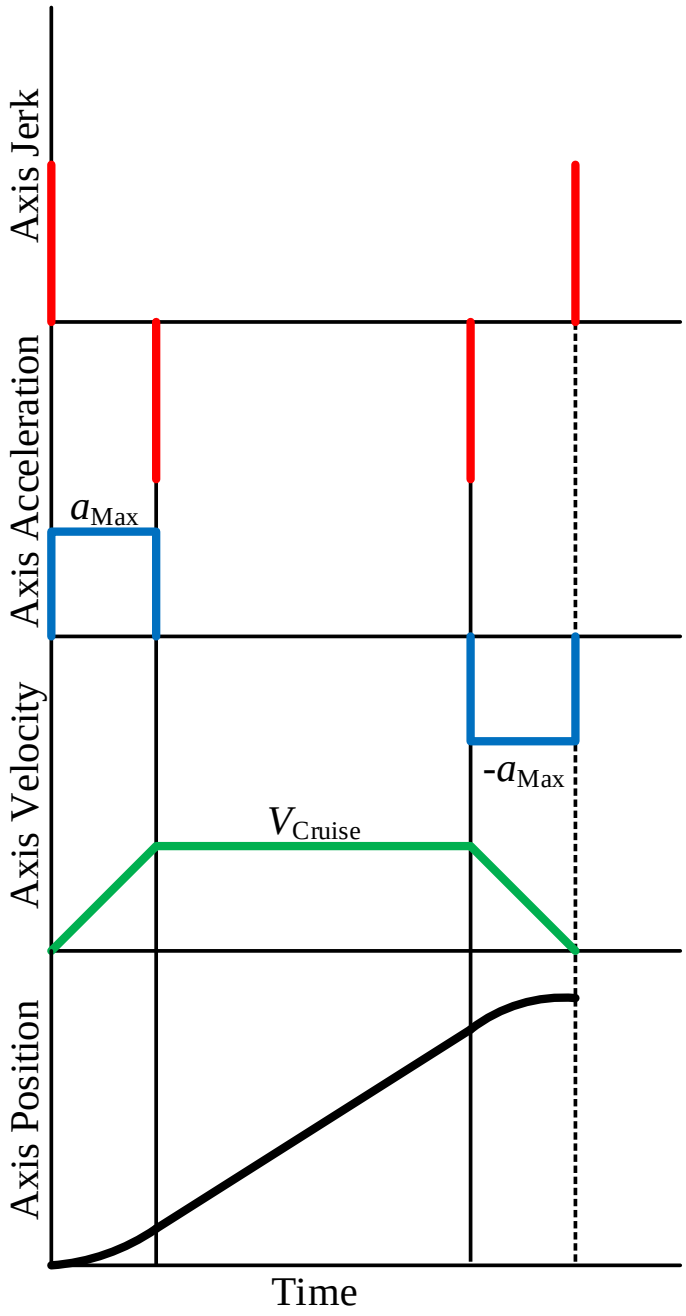


Figure 1.10: Trapezoidal Velocity Profile

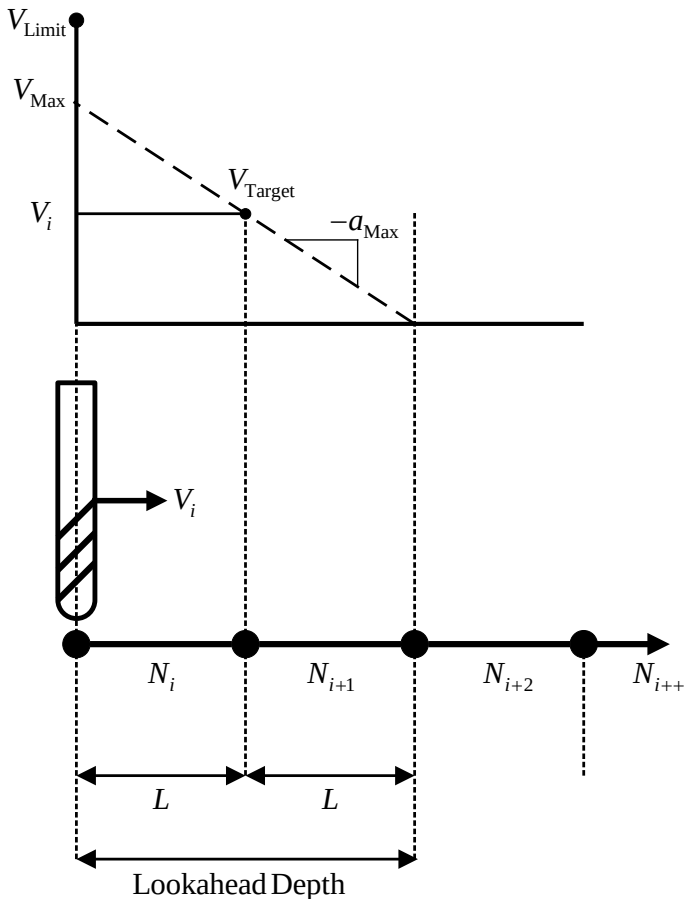


Figure 1.11: Single-Segment Lookahead Trajectory Planning

and uses the kinematic limits of the machine tool to determine when to begin deceleration for cornering or stopping [103, 104]. Due to the finite size of the lookahead buffer, the machine tool must assume that, in the worst case, the final block in the buffer is the last line of the toolpath and the machine should cease all motion at the end of that block [105]. This concept is illustrated in Figure 1.11. The cutting tool is traveling along a path composed of many small collinear movements at velocity V_i. The lookahead depth is one segment, and the maximum acceleration of the axis is $\pm a_{\text{Max}}$. If the length of the segment in the buffer is L, then the machine can only begin the block N_{i+1} at a velocity V_{Target} that will allow it to come to a complete stop in a distance L. Although the absolute maximum axis velocity imposed by the drive system V_{Limit} may be substantially larger than V_{Max}, the trajectory planner will not command a velocity higher than V_{Max} as the machine will not be able to stop by the end of the segment in the buffer. To avoid continuous accelerations and decelerations to V_{Target}, the machine will continue to travel at a constant V_i. It is apparent that, as the length of each movement in the lookahead buffer gets smaller, the maximum realizable velocity of the cutting tool along the path decreases. This can be problematic for very densely-sampled toolpaths, such as those derived from a voxel model, as will be shown in subsequent sections.

1.5.4 Interpolator

The trajectory planner populates a buffer of rough position and velocity commands that the tool must follow in order to machine a given toolpath without incurring substantial positional error. The trajectory planner is run on a realtime thread with period T_{TP} to fill a buffer which is subsequently read by an interpolator that runs on a different realtime thread with period $T_{\text{Interpolator}}$. In some implementations, the interpolator is neglected and the TP supplies points at high enough density to result in smooth motion. In most CNC systems, however, the interpolator further subdivides the points created by the trajectory planner by both fitting and interpolating along spline segments created between TP points. The

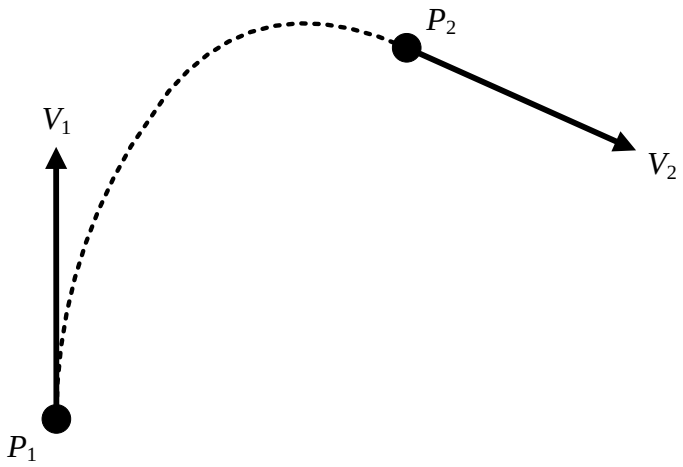


Figure 1.12: Cubic Interpolation

thread period of the interpolator is usually shorter than that of the TP because interpolation incurs less computational expense than trajectory planning and is thus easier to implement at a higher realtime rate [106]. The most common form of interpolation is cubic spline interpolation, where a Hermite problem is solved to fit a cubic polynomial between two target positions and velocity vectors from the TP. The cubic segment is then interpolated at $T_{\text{Interpolator}}$ to create position commands along the segment. By sampling both positions and velocities at the beginning and end of the cubic segment, feedrate can be smoothly interpolated along the segment without requiring step changes in velocity or acceleration. An example of cubic interpolation between TP points P_1 and P_2 with respective velocity vectors V_1 and V_2 is shown in Figure 1.12. While cubic interpolation is relatively simple, this approach does require step changes in jerk, which are not realizable and can lead to excessive machine vibration [107]. To remedy this problem, newer implementations use higher-order interpolators, such as those using quantic splines, that interpolate position, velocity and acceleration. An area of active research is the development of interpolators where constant arc length parameterization can be performed to eliminate fluctuations in feedrate over the toolpath [108].

1.5.5 Servo Controllers

The final step in creating machine motion from a G-Code toolpath is to output the interpolated points to the servo loops for each axis. Each axis is typically driven by a rotary servomotor coupled to some form of motion conversion mechanism, which could be a ball screw for translational axes or a harmonic drive for rotary axes. The servomotor provides actuation torque to accelerate the axis and is also equipped with some form of feedback device to measure either the shaft position of the motor or the position of the axis itself. Typically, this device is an encoder that provides position feedback. Most modern machine tools employ permanent magnet synchronous machines (PMSMs) for the axis drive motors that are driven by 3-phase power electronic inverters that supply drive waveforms of appropriate current and frequency to the motor windings. Velocity and position measurement of the motor can be accomplished by the use of tachogenerators and rotary or linear encoders as feedback devices. Commonly, a rotary encoder is placed on the motor shaft and the resulting position feedback signal is differentiated to obtain the motors rotational velocity. The control of the inverter itself is beyond the scope of this dissertation, but it is worth noting that modern digital servo drives contain the necessary control logic to align the stator and rotor magnetic fluxes such that maximum continuous torque can be produced over a wide speed range. As a result, the servomotor can be considered to provide constant torque over the relevant operating speed of the axis [109]. It cannot, however, provide an instantaneous change in torque due to the inductance of the motor windings that causes a nonzero rise time for winding current.

Depending on the structure of the motion control system, the actual feedback control of position may be performed on the same hardware as the trajectory planner and the interpolator, or it may be performed in the servo drive itself. The servo loops usually run at a higher frequency than the interpolator does because they must respond to rapidly changing dynamics to accurately control motor position. A high level block diagram of a typical joint space position control system is shown in Figure 1.13. Control is typically performed in joint

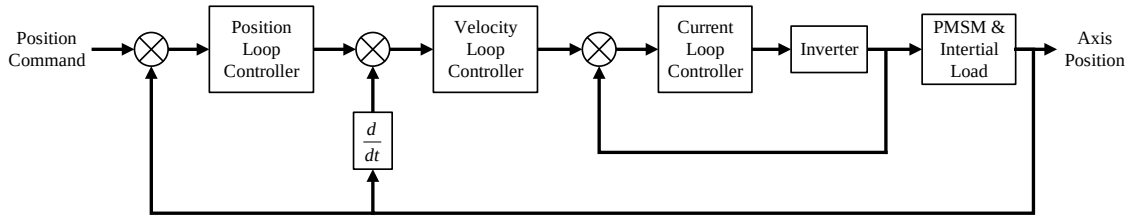


Figure 1.13: Typical Servo Control System

space so the controllers can be independent single-input single-output (SISO) systems. The interpolator commands position setpoints for the servo controller every $T_{\text{Interpolator}}$, and the servo loops track these setpoints with a period of T_{Servo}, also known as the servo rate. The controllers used in the servo loops are typically standard proportional-integral-derivative (PID) type controllers on industrial CNCs. For the purposes of this dissertation, the design of the control loops themselves is considered to be complete and they can be treated as a black box with the fine position command as an input and the actual axis position as an output.

1.6 Machine Tool Performance with High-Density G-Code

Current work has demonstrated that control of material removal rate using traditional G-Code is neither accurate nor predictable. Because of the large number of linear movements that the machine tool controller must read from the G-Code file, performance penalties have been observed in maintaining programmed tool velocity. These penalties are imposed by the nature of online trajectory planning, where the control system must be capable of handling deceleration phases of the toolpath without overshooting some positional error bound that can be set by the user [42]. The controller interprets the G-Code file at runtime, and thus does not have knowledge of the entire path before execution begins. Because of the finite depth of the lookahead buffer, the machine is forced to govern its maximum feedrate to a value that it can decelerate to zero at the end of the buffer. This behavior leads to a saturation of programmed tool velocity where, for a low programmed velocity, the

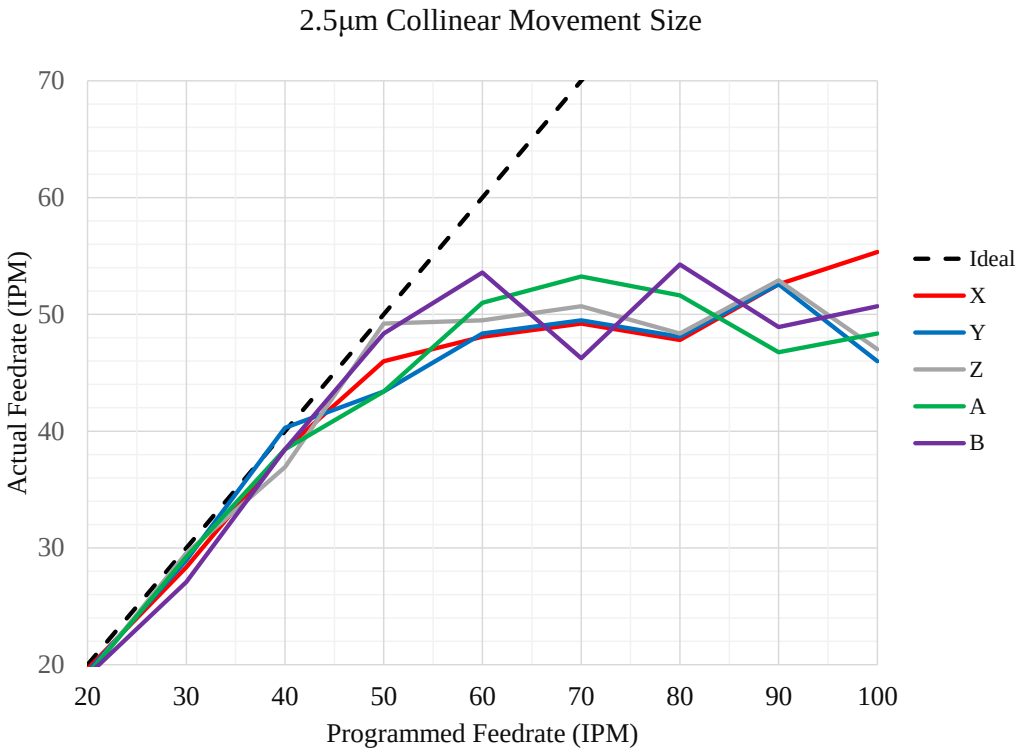


Figure 1.14: Feedrate Saturation of Mazak VCU-500A/5X

machine tool behaves as expected; however, for a high programmed velocity, the machine is not able to reach the desired feedrate.

Experimental results on a Mazak VCU-500A/5X 5-axis milling machine demonstrate this phenomenon. The maximum achievable feedrate is dependent upon the size of the linear movement commands that are provided to the machine. Figure 12 shows the actual feedrate of the machine, on a per-axis basis, when it is provided a densely sampled tool-path consisting of 10,000 sequential collinear movements that are each 0.0001” long. The programmed feedrate was increased by 10 inches per minute (IPM) for each consecurtive trial and the execution time of the toolpath was measured to determine the actual average feedrate during toolpath traversal. When using 0.0001” blocks, the level of feedrate saturation was determined to be approximately 50 IPM. This experiment was repeated for a block length of 0.001”, which revealed a saturation velocity of approximately 120 IPM. The observed saturation velocity is a function of the length of the movement segments; in other

words, the length of the linear region in Figure 12 is dependent on the lengths of the blocks within the buffer. This experiment was designed to determine if lackluster machine performance on dense toolpaths was due to the kinematic limits of the machine or some other reason. The control system on this machine, the Mazak SmoothX control, is marketed as one of the fastest and most capable machine tool controllers in the industry; however, even this control is unable to maintain programmed feedrate when following densely-sampled toolpaths. This behavior is problematic for applications that use toolpaths created from voxel models, as the manufacturing engineer must be capable of accurately controlling the feedrate of the tool in order to maintain consistent surface finish on the part. Additionally, control of MRR is not feasible if the machine tool cannot track the commanded toolpath at the programmed velocity.

1.7 Direct Machine Tool Servo Control Using Open-Architecture CNC Systems

The previous discussion on the state-of-the-art of machine tool motion control systems that rely on G-Code input reveals that a number of discretization intervals are present as a toolpath is converted from G-Code to servo loop setpoints. The coarsest level of the toolpath is the G-Code itself, in which movement lengths can range from microns to hundreds of millimeters. The trajectory planner discretizes the primitives in the movement queue temporally and assigns target velocities to each discretization point. The interpolator then commands the servo control loop at the interpolation rate using a spline fit between each trajectory point. It is clear from the discussion on toolpaths derived from voxel models that interpretation and trajectory planning for regularly spaced points along a voxel model is inefficient and leads to feedrate saturation.

Many researchers have attempted to address some of the deficiencies of standard G-Code for motion control of CNC machine tools. Minhat, et al. demonstrated an open CNC system based on IEC 61499 that was capable of interpreting and executing feature-based STEP-NC machining plans directly [110]. The features of the part model were parsed and

movement commands were sent directly to the axis motion control system using the trajectory planning strategies developed for the Enhanced Machine Controller (EMC). This system showed a novel application of an open CNC system for the machining of simple 2.5 axis features. Tsai, Farouki, and Feldman designed and implemented interpolators for Pythagorean Hodograph (PH) curves using an open 3-axis CNC system whose control software, known simply as OpenCNC, enabled easier implementation of a custom interpolator than would be possible with standard proprietary control systems [111]. This system showed that performance slowdowns that are experienced with small G-Code segments can be attributed to the path planning and interpolation strategies used by the control system, and the use of a PH interpolator allowed the machine to traverse a complex 3-axis path with greater speed than was previously possible. However, as this work points out, the usefulness of the interpolators is still limited by the proprietary nature of some aspects of the control system, even though it is marketed as open. Beaudaert, Lavernhe, and Tournier created a truly open CNC system, known as PREMIUM-OpenCNC, in order to directly interpolate the trajectory of a ball-end milling tool on the surface of a part while simultaneously considering the dynamic constraints of the machine tool [112]. Instead of creating and interpolating trajectories in joint space as is typically done with industrial CNC machines, this work interpolated a 5-axis trajectory in the parametric space of the workpiece to eliminate approximation errors that are introduced in typical CAM-generated toolpaths. The interpolation scheme was demonstrated on the open CNC system and showed improvements in performance as compared to the current industrial gold standard Siemens 840D. The authors correctly point out that, even with the high performance and relatively open 840D controller, the implementation of this machine tool control technique still required the development of a custom, in-house CNC system. Many previous works have also explored exotic interpolation schemes for rough points generated by a trajectory planning algorithm; many of these schemes are concerned with ensuring velocity, acceleration, and jerk continuity between TP points. Erkorkmaz and Altintas designed a quintic spline interpolation

method that was implemented on top of a jerk-limited trajectory generation strategy [106]. This work demonstrated that the use of high-order spline interpolators running at the servo loop rate were able to accurately follow the trapezoidal acceleration profiles created by the TP. Chang, Tsai, and Kuo proposed the implementation of a realtime NURBS interpolator in the servo control loop that would directly follow cutter contact (CC) points along a surface when provided with control points, a knot vector, and weights to define the surface [113].

Prior work has also explored the area of direct input to servo loops from rudimentary CAM software for machine tool motion control systems to realize gains in both toolpath execution time and accuracy. Chiu and Tomizuka demonstrated a task frame oriented approach in which the machine tool dynamics were transformed to the task frame and both tangential and normal contouring errors were controlled directly; this system relied on direct input to a servo control system and was capable of controlling tool position with higher bandwidth than was possible with typical G-Code [114]. Li, Zhao, and Ding implemented sliding mode contouring control on a 5-axis machine tool using direct control of the servo loops for each axis; toolpaths were planned and interpolated offline, and the post-interpolated data were fed directly to the sliding mode controllers to realize an improvement in contouring error over traditional PID-type position control [115]. Other researchers have performed machining from voxel models to show the feasibility of voxel-based CAM. Tarbutton, et al., demonstrated 3-axis machining of freeform surfaces using toolpaths created from a voxel model with 0.26mm voxels. In order to guide the tool along the path, G-Code was created from linear segments between voxels [11]. Lynn, et al., created mechanisms capable of relative motion after machining from a single piece of stock. These mechanisms were designed and converted into voxel models using the SculptPrint CAM system and the machine was controlled with G-Code using inverse-time feed [1]. To date, however, no research has explored the area of direct control of CNC servo loops from voxel-based CAM software.

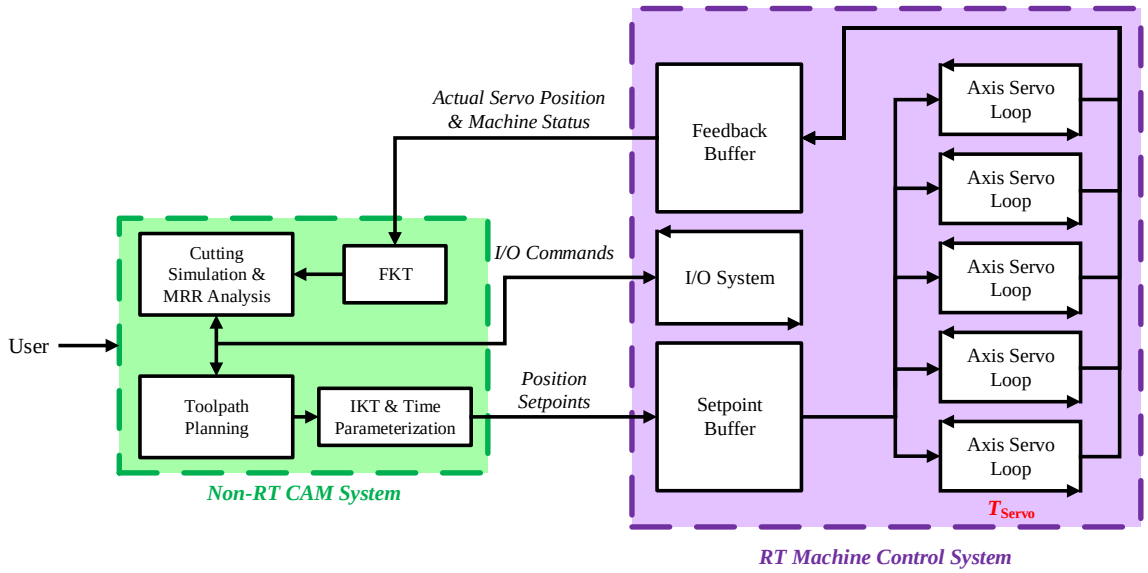


Figure 1.15: Direct Servo Control System Architecture

1.8 Direct Servo Control from Voxel-Based CAM

Instead of the typical interpretation, online trajectory planning, and interpolation stages necessary for the execution of typical G-Code programs, direct servo control uses preprocessing and offline trajectory planning from within the CAM system. In the present case, SculptPrint will determine servo loop position setpoints in joint space and feed them to a buffer that the axis position controllers will read at the servo loop closure rate of T_{Servo}. The control system will then feed back the position of each axis to the CAM system through a separate buffer that is populated by the control loops at the servo rate. By controlling the relative command position difference between each setpoint, arbitrary positional derivative profiles can be commanded on a per-axis basis. A block diagram for this system architecture is shown in Figure 1.15. Use of the IKT presented in Equation 1.5 will enable the control of tool space positions from the servo loop setpoints. SculptPrint will feed target position and time commands into a buffer thread set up in the motion control system, and the machine tool will then drive the axes to the positions specified in the buffer using a PID controller for each axis.

1.8.1 Trajectory Generation from MRR Data

The creation of tool center point locations and tool orientations for each step in a toolpath provides the necessary axis positional information to realize a toolpath, but velocities, accelerations, and higher positional derivatives still need to be assigned to each step along the path for it to be executed by the machine tool. The machine tool control architecture developed in this work will enable the following technique to determine servo loop setpoints and target arrival times at each voxel center. The time increment for a given step can be calculated by equating Equations 3 and 4 and substituting Equation 2 to yield

$$\Delta t_{i \to i+1} \geq \frac{\sum_{i} dv \times s^3}{\mathrm{MRR}_{\mathrm{Limit}}} \tag{1.8}$$

where Δt is the time required to complete the step if the instantaneous MRR is to be maintained below the tools $\mathrm{MRR}_{\mathrm{Limit}}$. A 2-dimensional example of MRR calculation over a step is shown in Figure 1.16. The cutting tool, whose envelope is denoted as C, is initially at point P_1 with tool axis vector $\mathbf{v}_1$. The movement direction vector, $\mathbf{V}_1$, denotes the distance and direction that the centerpoint of the tool will traverse during the step. The tool will complete the step once it has arrived at point P_2 with tool axis vector $\mathbf{v}_2$. Before reaching P_1, the tool has completed the path shown by the dashed line and removed the shaded red voxels. While travelling along $\mathbf{V}_1$, the cutter envelope C will contact the four shaded gray voxels of side length s. Equation 1.8 can therefore be evaluated as

$$\Delta t_{1 \to 2} \geq \frac{4s^3}{\mathrm{MRR}_{\mathrm{Limit}}} \tag{1.9}$$

If this process is repeated for every step along a given toolpath, a sequence of target tool center point positions, tool axis vectors, and arrival times will be obtained for the entire trajectory. By using the inverse kinematic transformation given in Equation 1.5, the necessary servo loop setpoints for the axis position controllers can be obtained. By recording the

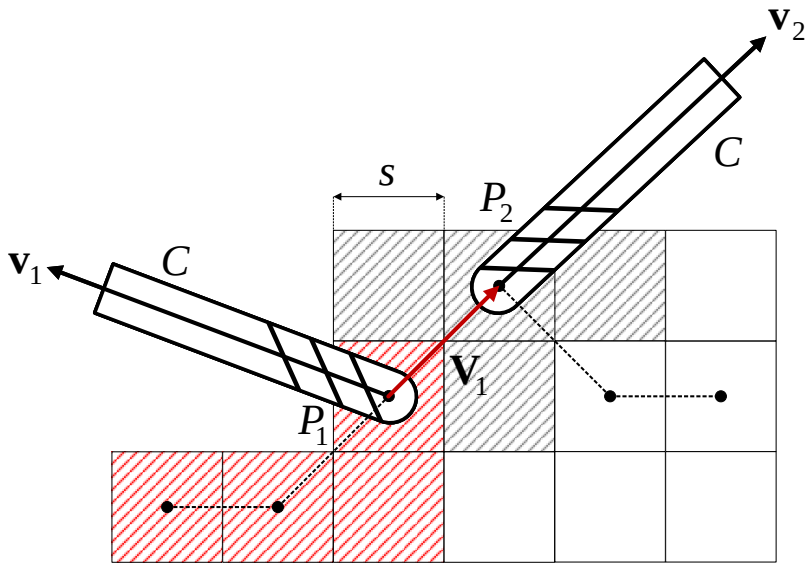


Figure 1.16: Voxel Removal Along a Step

actual axis positions at every servo update and passing the points through the FKT for the machine, the actually-executed path can be evaluated in the workpiece coordinate system.

The commanded arrival times must not only be constrained to not violate the maximum MRR for a given tool, but also they must be formulated to respect the kinematic limits of the machine as described in Equations 1.7. Equations 1.10 give these constraints, formulated up until the jerk limit of the machine,

$$\Delta t > 0 \tag{1.10a}$$

$$\frac{\Delta P}{\Delta t_{i\to i+1}} \leq V_{\text{Max}} \tag{1.10b}$$

$$\frac{\Delta^2 P}{\Delta t_{i\to i+1}{}^2} \leq A_{\text{Max}} \tag{1.10c}$$

$$\frac{\Delta^3 P}{\Delta t_{i\to i+1}{}^3} \leq J_{\text{Max}} \tag{1.10d}$$

where ΔP represents the change in position of an axis over a given time increment Δt and the n^{th} exponent denotes the n^{th} discrete derivative. These limits on Δt can be used to formulate a constrained optimization problem in which some cost function is minimized over the course of the path and suitable arrival time commands are assigned using the results

of the optimization. The most obvious optimization would be to minimize machining time while not violating the MRR limit or the kinematic constraints of the machine tool.

Careful consideration must be paid to the fact that control of the machine is performed discretely in the time domain, while the part model and resulting toolpath is discretized spatially. As a result, the position loop setpoints in the command buffer will most likely not correspond exactly to the servo rate. Depending on the size of voxels used in the part model and the time increment assigned to each step of the toolpath, the axes may reach the target positions between servo cycles. Additionally, the axis movements required for each setpoint may be large enough for the axes to traverse the required distance in one servo update. Three distinct possibilities exist for coordinating spatially discrete points with a time domain control system:

1. The movement distance between setpoints and the commanded time increment result in a high enough velocity that the machine will reach the setpoint before the next servo update.

2. The movement distance between setpoints and the commanded time increment result in a velocity such that the machine will exactly reach the setpoints when the next servo update begins.

3. The movement distance between setpoints and the commanded time increment result in a velocity such that the machine will reach the setpoints between servo updates.

For the control implementation developed for this work, trajectories are generated using a time-optimal path planning strategy developed in [116] and [117], which considers only the kinematic limits of the machine tool axes. However, the machine tool software and hardware developed in this research provide the foundational capability needed to implement MRR-constrained toolpaths.

1.8.2 Goals of this Research

This dissertation aims to develop and evaluate a general machine tool control framework that is capable of executing arbitrary positional derivative profiles that are generated by an external trajectory planning system. Realization of the control system will provide the groundwork necessary to control tool position in the workpiece coordinate system using both kinematic constraints of the machine tool axes and cutting tool MRR constraints. The direct control system requires development and construction of numerous pieces of machine tool hardware and software, which will be described in detail. A successful implementation of the direct control idea will not only allow SculptPrint to directly write the servo loop setpoints for each axis of the machine tool at the loop closure rate, but also it will enable analysis of the actually-realized trajectory along the path by recording the true axis positions at every servo period.

Contributions

The development and evaluation of this system will enable the following additional functionality that is not present in industrial CNC systems that use standard G-Code:

1. The CAM system will have more complete control over axis motion than is currently possible with standard G-Code.

2. Near-realtime feedback of actual axis positional derivative information will be available in the CAM system to enable MRR analysis by manufacturing engineers.

3. Material removal rate can be accurately controlled on a voxel-by-voxel basis using trajectories that are generated with MRR constraints.

4. Arbitrary positional derivative profiles will be realizable directly from the CAM system, which will enable future research into toolpaths that control jerk, jerk rate, and higher positional derivatives.

Evaluation Metrics

The measure of success in this implementation will be the accuracy with which the trajectory can be controlled at each step along the toolpath. This is an important consideration in high complexity machining operations, and complete control of trajectory is not effectively realizable with current machine tool programming practices. A collection of experimental toolpaths will be developed and executed on the completed machine tool control system, and the axis position and velocity progressions (in joint space) will be measured by various means and compared with the planned trajectory. The FKT will be used to convert the joint space positions to the corresponding tool space positions, which will be compared to the commanded trajectory to determine positional error at the workpiece. Additionally, the temporal performance of various critical aspects of the software developed for this dissertation will be performed to determine if the direct control system is capable of executing the high density paths that are generated from the voxel-based CAM system. Finally, execution time of the experimental toolpaths will be compared between the direct control system and standard G-Code programming to evaluate whether the direct control system is capable of addressing the problems of feedrate saturation described in Figure 1.14.

Research Questions Addressed

The results and analysis presented in this work will address a range of research questions concerning fundamental aspects of CNC machine tool control. These questions, which are enumerated below, will be revisited in the final chapter of this dissertation using knowledge gained during development and experimentation.

1. How well can the near-RT components (such as the setpoint buffer) of a machine tool machine be controlled from a non-RT CAM system?

2. To ease computational load on the embedded system that performs the physical motion control of the machine, it would be ideal to offload heavier computational tasks

to a more powerful, but not necessarily realtime, external platform. How can CC be leveraged in a machine tool control application?

3. How densely can toolpaths be described to a motion control system with an offline trajectory planning stage without affecting programmed tool velocity?

4. Can the problem of velocity saturation due to finite lookahead depth be solved by planning trajectories in the external TP instead of online planning as is done in commercial systems?

5. If process feedback to the CAM system is implemented within the motion controller, how can the resulting data be used for iterative trajectory optimization?

6. How accurately can the planned trajectory be maintained by the direct control system, and is this accuracy limit something repeatable that can be compensated for?

Assumptions

A number of assumptions are made is this dissertation, each of which is enumerated below:

1. The MRR limit obtained from Equation 1.4 is the maximum amount that the cutting tool is capable of and will never result in tool breakage.

2. The kinematic limits of the machine tool given in Equations 7a-c are accurate, available from the machine tool manufacturer, and are not affected by current axis motion or cutting conditions.

3. The position measurement obtained from the servo controllers is perfectly accurate.

4. The servo control system can be treated as a black box that simply accepts position commands and outputs the actual axis positions.

5. Deflection of the cutting tool and machine structure is negligible, the cutting tool is perfectly sharp, and the MRR computed from actual axis positions is perfectly accurate.

6. No geometric errors exist in the machine tool structure, implying that application of the FKT to the joint positions results in perfectly accurate tool space positions

7. Numerical errors in trajectory generation are non-existent, and thus numerical differentiation of commanded position setpoints results in exact positional derivative profiles

Limitations

The successful demonstration of the proposed approach will have a number of limitations, each of which is enumerated below:

1. Hard realtime control of the machine tool from the CAM system will not be possible, as SculptPrint will not be run in an RTOS.

2. In-process speed override of the machine will not be possible without near-realtime trajectory replanning, as the trajectories are not planned online. However, this could be addressed by asking the TP for trajectory recomputation with different kinematic limits.

3. Network latency between the CAM system and the machine tool controller can be problematic and cause an exhaustion of the setpoint buffer, resulting in unpredictable or unexpected motion of the machine axes.

CHAPTER 2
THE RESEARCH MACHINE TOOL SYSTEM

The experimental setup for developing and implementing the direct servo control system required the design and construction of numerous mechanical and electrical subsystems to support research activities on the machine. This section describes each relevant piece of hardware that was developed over the course of this research. The main components are a central control computer, a 5-axis machine tool structure, a realtime machine tool control system, a data acquisition system, and associated motor control systems.

2.1 Control Computer

A central control computer is responsible for running the SculptPrint CAM system, communicating with the machine tool controller, and collecting process data during machining. The control computer, shown in Figure 2.1, is equipped with an Intel Xeon E5-1650 CPU, 128GB of memory, and an NVIDIA Quadro M5000 GPU. The local operating sytstem of the control computer was chosen as Windows 10 to enable native support of SculptPrint; in addition, a collection of Debian virtual machines were created on the control computer to enable software development and debugging in a Linux environment. The hardware of the control computer was chosen to provide enough computing power for both toolpath planning and control of the machine tool.

2.2 Machine Tool Structure

The CNC machine tool iself is based on a hobby-grade 5-axis CNC machine tool known as the PocketNC. The PocketNC began as a crowdfunded project that eventually spawned a small company, The PocketNC Company, that manufactures the machines and sells them

Figure 2.1: Construction of Control Computer

for approximately $4000 each. A stock PocketNC is shown in Figure 2.2.

The machine tool used for the experimental platform is a highly-modified PocketNC that is equipped with additional mechanical and electrical systems that are not present on the stock PocketNC. To accommodate current and future upgrades, a frame was constructed to convert the PocketNC from a horizontal spindle machine to a vertical spindle machine. The frame was CNC machined from 6061 aluminum plate and features extruded aluminum rails for mounting accessories. A CAD representation of the frame assembly is shown in Figure 2.3, and the fully-assembled machine tool structure is shown in Figure 2.4. Each axis of the machine is powered by a stepper motor (in the case of the translational axes, a single stepper motor drives a leadscrew to move the axis; in the case of the rotational axes, a pair of stepper motors drive a timing belt to move the axis), and each axis is equipped with a 2000 count-per-revolution optical rotary encoder. The stepper motors are driven by TI DRV8825 stepper driver integrated circuits (ICs). The kinematic limits of the machine tool are given in Table 2.1.

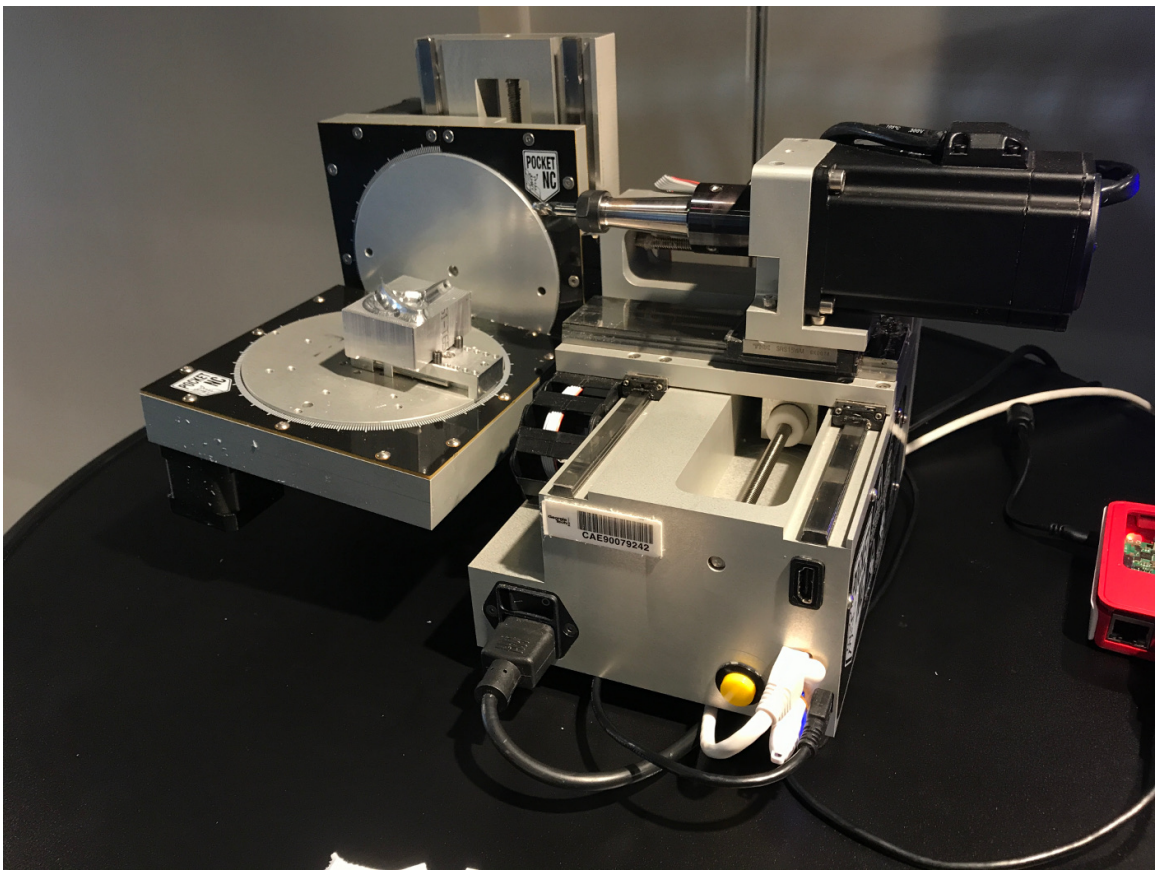


Figure 2.2: Unmodified PocketNC

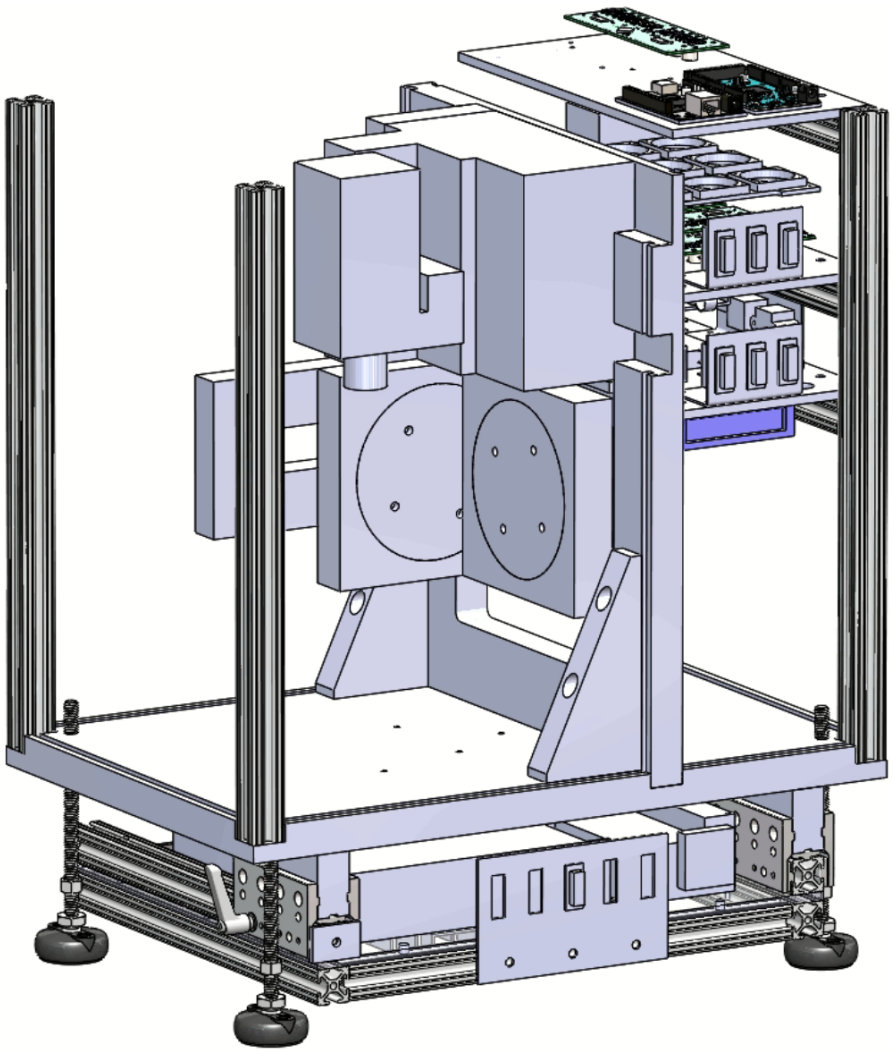


Figure 2.3: CAD Model of Machine Assembly

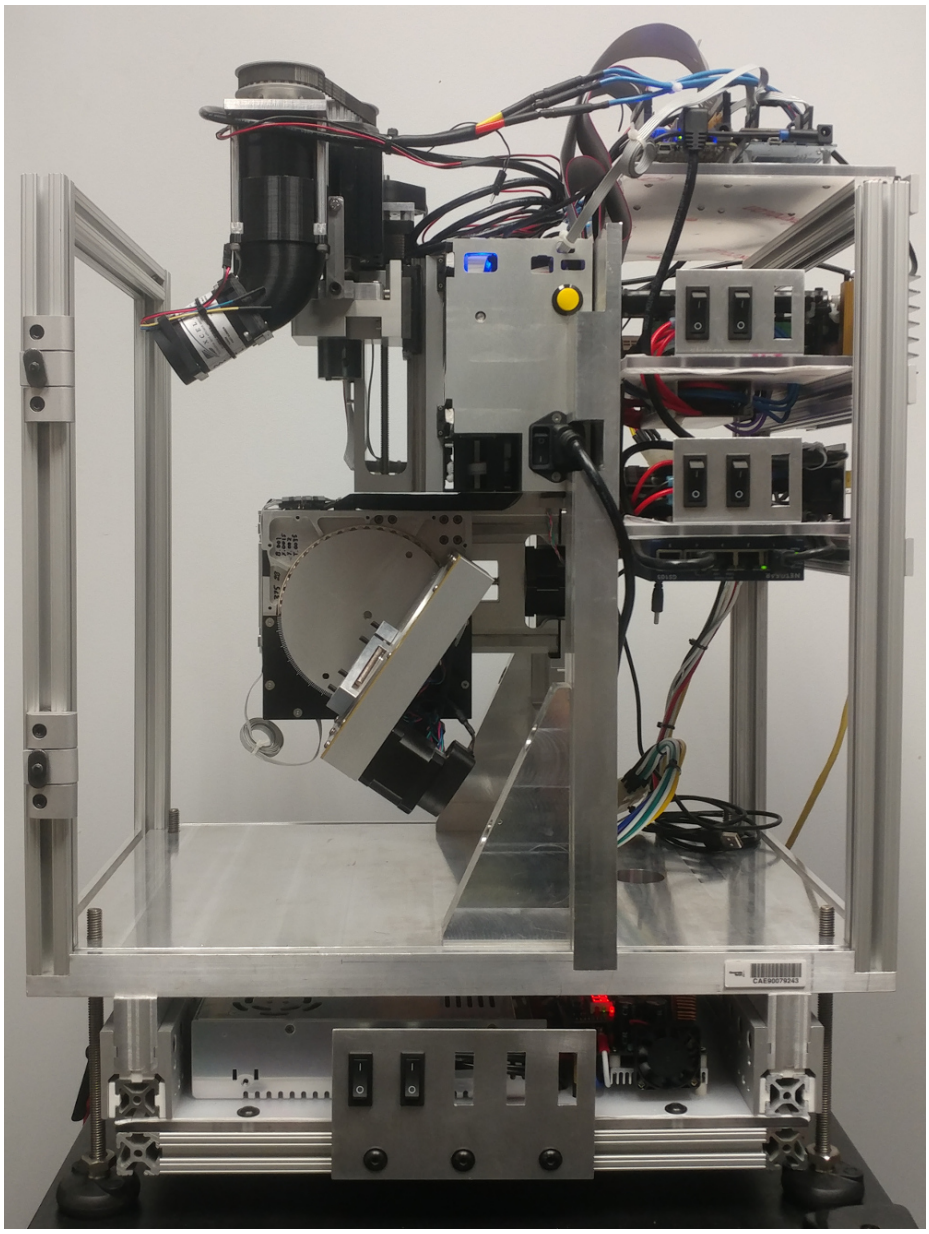


Figure 2.4: Completed Machine Tool Assembly

Translational Axes	Travel Limits (mm)	Velocity Limit (mm/s)	Acceleration Limit (mm/s^2)
X	-44.45 +64.77	16.93	762
Y	-52.07 +74.93	16.93	762
Z	-87.63 +2.54	16.93	762

Rotary Axes	Travel Limits (deg)	Velocity Limit (deg/s)	Acceleration Limit (deg/s^2)
A	-5 95	10	1500
B	Continuous Rotation	10	1500

Table 2.1: Kinematic Limits of Research Machine Tool

2.3 Machine Tool Control System

The control system for the machine tool is based on a Beaglebone Black (BBB) single board computer that runs a a modified version of the Debian Linux distribution. Control of the machine axes and input/output (IO) systems is performed using an open-source machine control software system known as Machinekit. Machinekit is a spinoff of the LinuxCNC project, which is in turn a spinoff of the Enhanced Machine Controller (EMC) project developed by the National Institute of Standards and Technology (NIST) [93]. The BBB, shown in Figure 2.5, is powered by an ARM Cortex-A8 processor and is equipped with a programmable realtime unit (PRU) for high speed realtime control tasks. The BBB communicates with the main control computer using an Ethernet connection.

2.4 Power Distribution

Electrical power is provided to the various subsystems of the machine tool from a centralized power converter panel that is housed beneath the machine structure. All power consumed by the machine originates from 120V AC wall power. The main DC bus is sup-

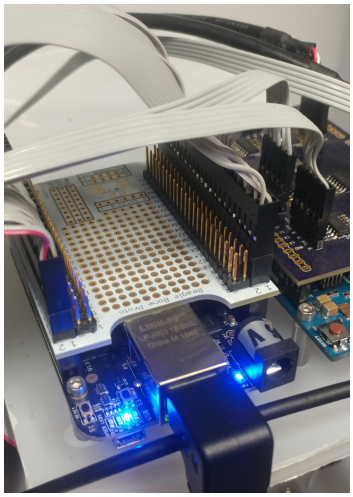


Figure 2.5: Beaglebone Black Running Machinekit

plied by two 12V 30A AC-DC power supplies connected in series. An 8A adjustable buck converter, supplied by the main DC bus, is set to 5V and used to power 5V accessories (such as USB power); a 12A adjustable buck converter, also supplied by the main DC bus, is set to 12V and used to power 12V accessories (such as cooling fans and an Ethernet switch); and a 15A adjustable boost converter, set to 48V, is used to power the spindle drive. The power distribution panel is shown in Figure 2.6. Table 2.2 shows the voltages used for the various electrical components of the machine tool.

Input Voltage	**Component**
120 VAC	PocketNC 120VAC to 24VDC Power Supplies
48 VDC	Spindle Drive
24 VDC	24V to 48V Boost Converter 24V to 12V Buck Converter 24V to 5V Buck Converter
12 VDC	Ethernet Switch Spindle Drive Fans Spindle Ducted Fan SSR Control Voltage
5VDC	USB Hub

Table 2.2: Research Machine Tool Power Supply Configuration

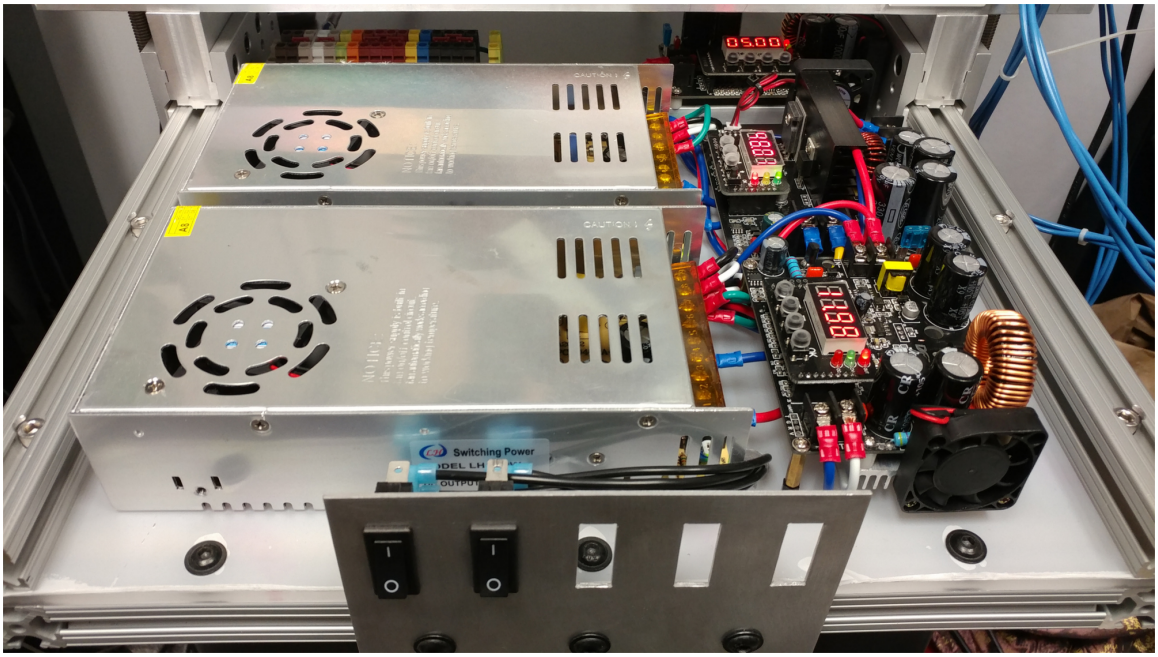


Figure 2.6: Main Power Distribution Panel

2.5 Communication

Two forms of communication are used to interface with the machine tool: Ethernet and USB. The Ethernet connection is used to communicate with the BBB, and the USB connection is used to communicate with both the encoder interface microcontroller and the spindle drive. An Ethernet switch and a USB hub, in addition to terminal blocks and the associated wiring for power distribution to parts of the machine mounted away from the central power converter panel, were attached to a panel that is mounted on the aluminum framing rails used on the machine frame. This auxiliary power distribution and communication panel is shown in Figure 2.7. Power to the communication panel is provided by a connectorized cable that is connected to the main power converter subsystem described in Section 2.4

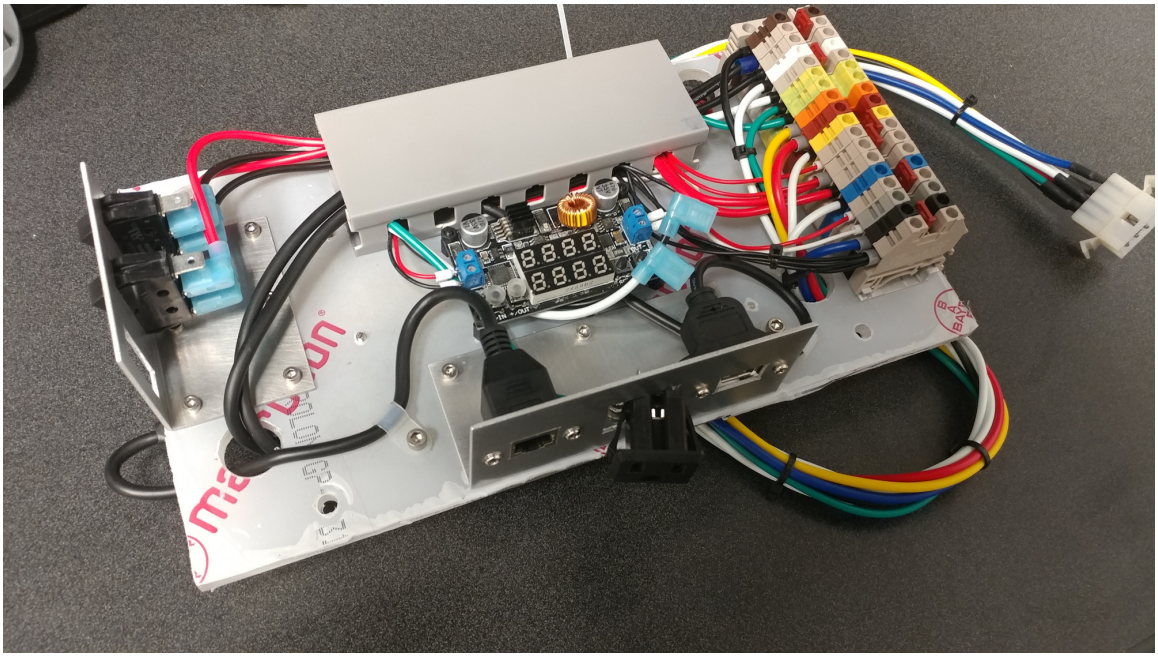
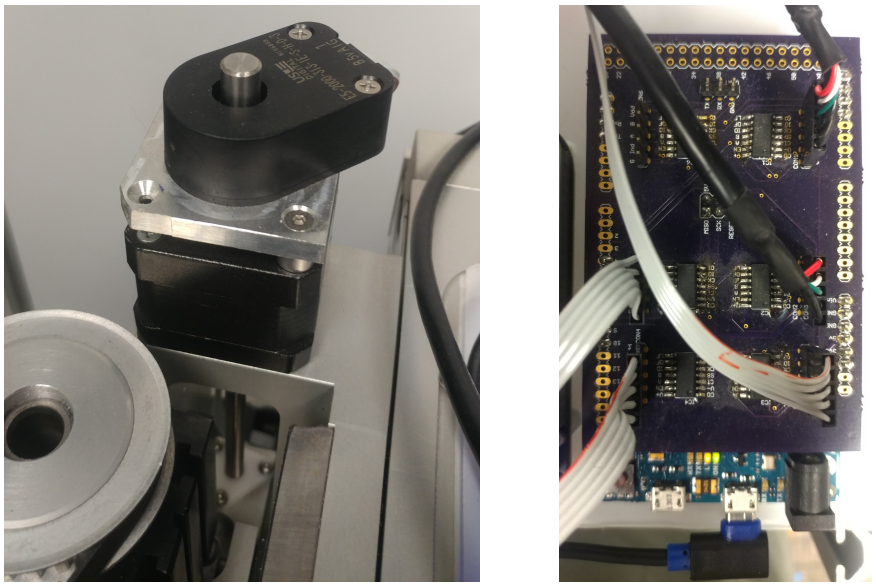


Figure 2.7: Communication and Auxiliary Power Distribution Panel

2.6 External Data Acquisition System

The experimental platform is also equipped with an Arduino Due microcontroller board that interfaces with rotary encoders that were installed on each machine axis. Each encoder is connected to an LSI Computer Systems LS7366R-S 32-bit quadrature decoder, and each decoder is mounted on a custom printed circuit board (PCB) that was developed and manufactured for this dissertation. The encoder interface board was designed to mount on top of the Due, and is shown fully assembled in 2.8(b). The Due communicates with quadrature counters using a serial peripheral interface (SPI) connection. The Due communicates with the main control computer using a USB-RS232 serial connection. The rotary encoder and accompanying mounting hardware developed for the Z-axis is shown in Figure 2.8(a); this encoder is connected to the quadrature decoder board with a ribbon cable.



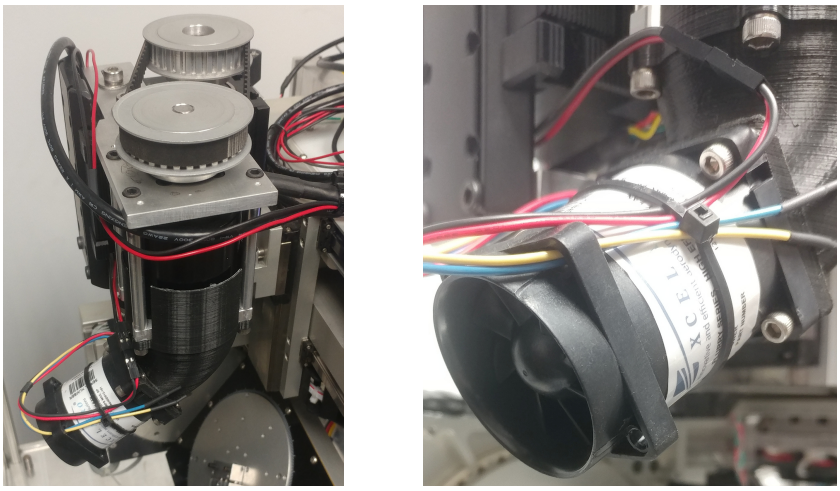
(a) Z-Axis Encoder

(b) External Quadrature Decoder Board

Figure 2.8: Axis Encoder and Quadrature Decoder PCB

2.7 Spindle

The spindle of the machine is powered by a D5065 270kV three-phase brushless motor that is controlled by a servo controller known as the ODrive. The spindle motor is equipped with a 8192 count capacitive encoder and drives the spindle itself through a timing belt. Motor cooling is provided by a 12V brushless axial ducted fan that is powered by the buck converter described in section 2.4. The spindle cooling fan is shown in Figure 2.9(b). The ODrive communicates with the main control computer using a USB connection. The spindle assembly is shown in Figure 2.9(a), and the ODrive used to control the spindle position and velocity is shown in Figure 2.10. DC bus power to the ODrive is switched with a Crydom 100A solid state relay (SSR).



(a) Spindle Assembly (b) Spindle Cooling Fan

Figure 2.9: Spindle Assembly and Ducted Cooling Fan

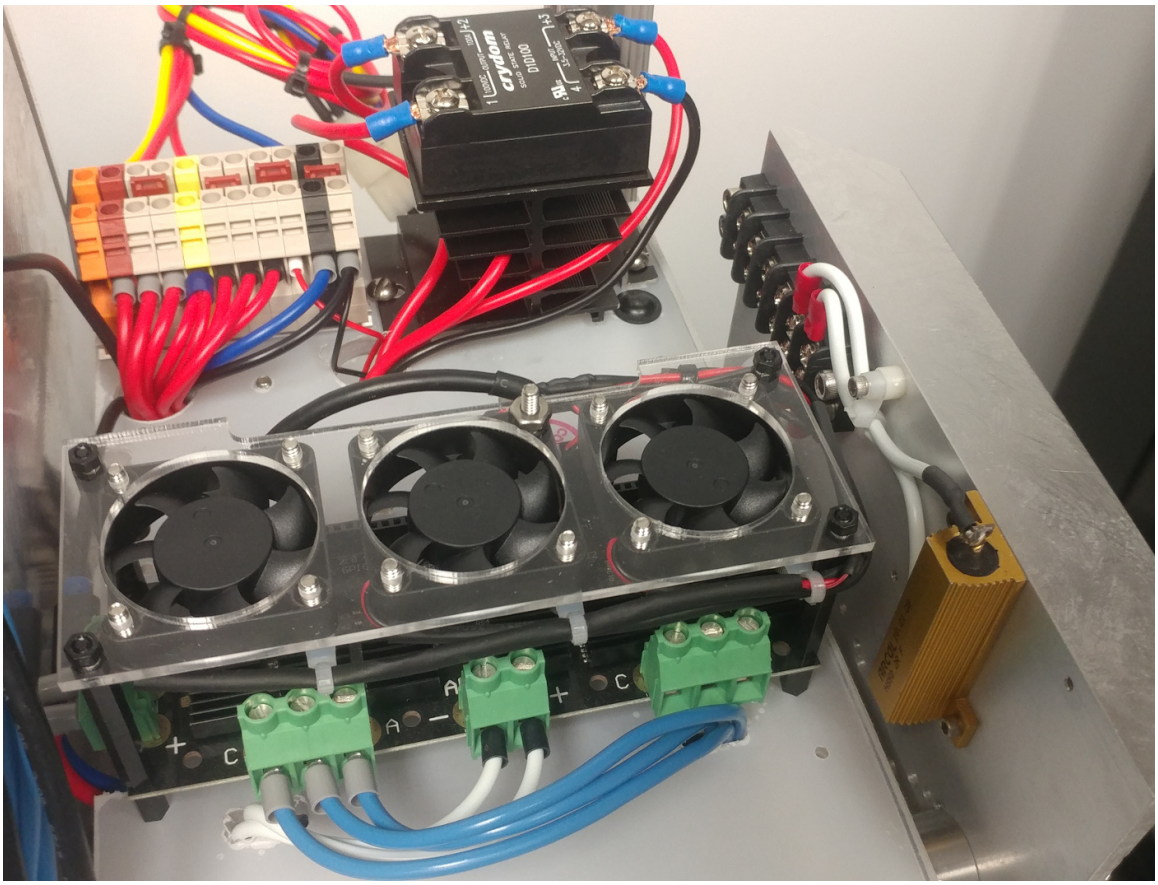


Figure 2.10: ODrive Panel, Wiring, and SSR

CHAPTER 3
THE CAM-CNC INTERFACE SOFTWARE

The bulk of the work for this dissertation was devoted to software development, both for the main control computer and for the BBB. Changes to the core Machinekit kernel were implemented to expose the axis servo loops to the CAM system, non-RT communication code was written for the BBB, and an interface application was developed for the control computer to enable SculptPrint to communicate effectively with the BBB. The combination of these software elements creates an interface layer between SculptPrint and the motion control subsystem of the machine tool that enables the high speed data exchange necessary to realize the direct control system.

3.1 Xenomai

The realtime system used to control the motion axes of the machine is an integral part of the direct servo control system. As described in Chapter 2, the Machinekit project was used as a basis for the software of the control system that was implemented on the BBB. Machinekit is a completely open-source software CNC system that uses a realtime operating system (RTOS) to ensure consistent performance of critical motion and I/O control tasks. For this dissertation, the Xenomai subsystem was used to enable realtime control capability. Xenomai is an open-source realtime development framework that supports both user space and kernel space realtime tasks [118]. Xenomai relies the Cobalt co-kernel which runs alongside the traditional Linux kernel and handles high-priority realtime interrupts through an interrupt pipeline (I-Pipe) to the hardware abstraction layer (HAL). A simplified Xenomai system architecture is shown in Figure 3.1. The critical motion and IO control tasks performed by Machinekit are assigned to a realtime thread that is handled by Xenomai.

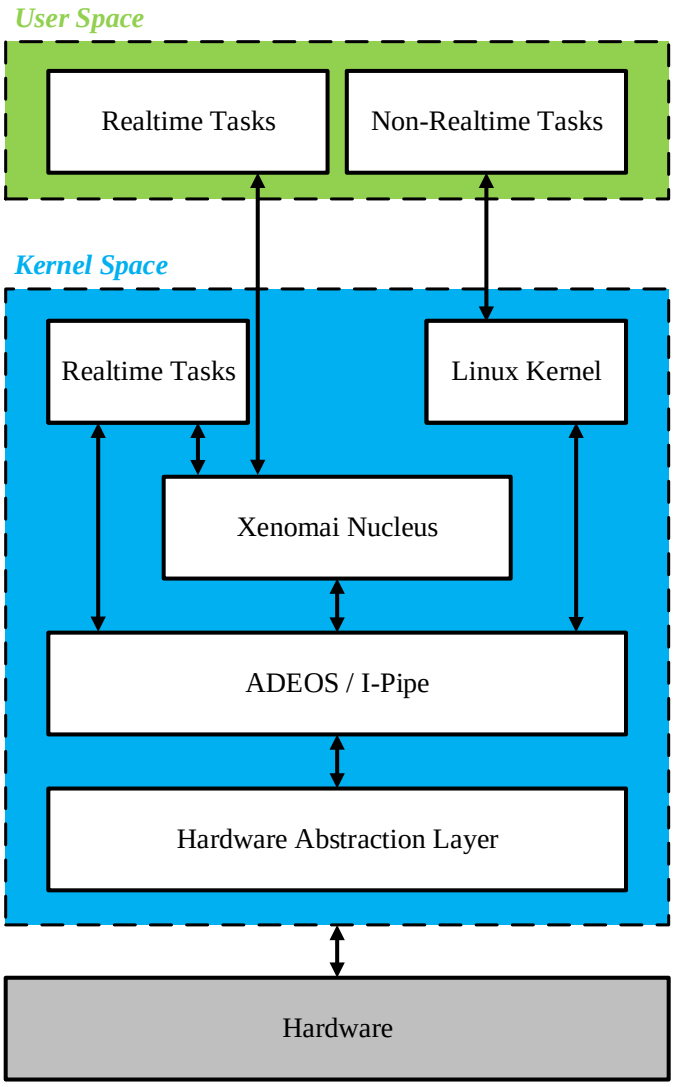


Figure 3.1: Simplified Xenomai Architecture

3.2 Embedded Control System

The Machinekit CNC system is designed to interpret typical G-Code using the RS-274 interpreter that is described in [43]. Online interpretation of G-Code requires trajectory planning and interpolation to be performed on the realtime subsystem, which is not required for the system developed in this dissertation. Instead, trajectories are planned and interpolated within the CAM system, and the resulting servo position commands are fed directly into Machinekit's servo loops. To realize this functionality, a variety of modifications were made to core Machinekit code and the resulting source was cross-compiled for ARM and installed on the BBB. The codebase for the directly controllable version of Machinekit is maintained in the author's Github repository.

Machinekit consists of a multi-process architecture where multiple independent processes are designated to handle different tasks and communicate with one another through a shared memory area. In this dissertation, all of the necessary Machinekit software modules are run on the embdedded machine control computer (the BBB). The Machinekit processes that are relevant to this research are the task executor (EMCTASK) and the motion controller (EMCMOT). In the stock Machinekit code, the task executor is a non-realtime component that accepts motion commands that are interpreted from a G-Code file or network connection and passes them to the realtime motion controller process. Commands are passed as neutral messaging language (NML) messages through a shared memory area that is accessible to both the task executor and the motion controller. The trajectory planner and interpolators are part of the motion controller process, and populate buffers of servo position setpoints (trajectory buffers) from commands that are received from the task executor. Communication with the outside world is performed using a non-realtime C++ application known as the Machinekit Remote Shell, which handles network communication with the CAM system.

3.2.1 Realtime Components

The ultimate goal of the realtime components of the direct control architecture is to realize motion of the axes of the machine tool. To ensure adequate motion performance, the motion controller is attached to a Xenomai RT_Task that is run every millisecond. This task, known as the servo thread, performs all motion and safety-critical functions at every millisecond update. In a typical CNC machine control system (and also in the stock Machinekit code), the realtime system is responsible for fitting and interpolating splines through critical motion points as defined by the G-Code program (e.g., start and end points of moves, points along circular arcs, etc.). The trajectory planner determines target velocity vectors at the critical points by accounting for machine kinematic limits, and then fits appropriate splines to go through the critical points and respect the desired velocity vectors. The interpolators then interpolate the splines by sampling them at the servo loop closure rate to determine position setpoints for the machine axis actuators. Such functionality is not required in the direct control architecture, as servo loop position setpoints are calculated by the non-realtime CAM system. As a result, the realtime component of the Machinekit codebase was modified to eliminate the G-Code interpreter and instead accept arrays of servo position commands from the task executor and pass them directly to the actuator servo loops. The task executor is commandable using the user-space shared memory area that was originally reserved for communication between the G-Code interpreter and the task executor. The task executor thus serves as an application programming interface (API) to the trajectory buffers in Machinekit.

Servo Controllers

The servo controllers for the experimental platform are implemented in software and utilize the HAL to command BBB hardware. The servo loops for the experimental platform do not follow the typical servo loop architecture for brushless PMSMs shown in Figure 1.13; rather, the axes are actuated by stepper motors which are driven by bipolar stepper driver

ICs and the control signal to each driver is simply a pulse train from the BBB's PRU. Positional error P_{Error} is calculated at every servo update by Equation 3.1,

$$P_{\text{Error}} = k_{\text{Axis}}(S_{\text{Required}} - S_{\text{Actual}}) \tag{3.1}$$

where S_{Desired} is the number of steps that were commanded at the completion of the last servo update, S_{Actual} is the actual number of steps that have been generated since the last servo update, and k_{Axis} is a scaling factor that depends on the mechanics of the drive system (e.g. the lead screw pitch in the case of linear axes, the reduction ratio in the case of rotary axes, the number of steps per revolution of the stepper motor, the number of microsteps configured in the stepper driver IC, etc.). Servo control is therefore performed in *joint space* for each axis independently. It is the responsibility of the trajectory planner to ensure that commanded servo loop setpoints will realize motion that is geometrically accurate in tool space. Equation 3.1 implicitly assumes that the axis actuator does not skip any steps that are provided by the driver IC, which is a valid assumption if torque load on the actuator, angular velocity, and angular acceleration are within the design specifications of the motor. Conformance to the kinematic limits in Equations 1.7 ensures that motor velocity and acceleration do not exceed the point at which steps will be skipped.

At every servo update, the servo control system computes current positional error for each axis and determines the suitable step frequency for each axis to reach the next position command at the beginning of the next servo update. The resulting step frequencies are then written to the HAL, which commands the PRU to output pulse trains to each stepper driver IC. The actual position commands are stored in a trajectory buffer that resides in a shared memory area that is accessible by both the motion controller and the task executor. The trajectory buffer is a first-in, first-out (FIFO) buffer where each entry in the buffer is a one-dimensional vector whose size is determined by the number of motion axes of the machine tool. For the experimental platform in question, each element has five entries, as

the machine tool has five motion axes; however, the software developed for this research is extensible to a machine tool with any number of motion axes. The size of the trajectory buffer is controllable by the user and its associated memory block is allocated at runtime. For this dissertation, a FIFO buffer with 2000 entries was used. During each servo update, the servo control system consumes the top entry of the FIFO and uses it as the position setpoint for the subsequent servo period. When using a servo update frequency of 1 kHz, the trajectory buffer can store two seconds of motion commands.

Servo Loggers

High frequency servo feedback information is not available from most commercial machine tools, and thus the data availability up the control chain is limited; the feedback only exists on the realtime control system for the purposes of closing the axis position loop. For the direct control architecture, availability of feedback information at the servo rate is essential to evaluating the motion of a toolpath.

To enable logging of high resolution servo position feedback, additional shared memory was allocated and written to on every servo update step with the current axis position. This shared memory area serves as a buffer for servo position readings that is regularly flushed to the CAM system. To control computational load on the realtime thread, a user-selectable subsampling rate was added and the feedback buffer size is a controllable parameter. The implementation for the servo logging system was written in C code that is run on the same Xenomai task as the other realtime responsibilities of the controller.

An alternating buffer configuration was used to ensure that feedback data is properly transmitted to the non-realtime side of the control system. While one buffer is being filled by the servo thread in the realtime process, the other buffer is waiting for retrieval by a separate thread in a non-realtime process. A pictoral description of this process is shown in Figure 3.2. Once the non-realtime thread has emptied a full buffer to the CAM system, it signals to the realtime thread that successful data transfer has occurred and a freshly

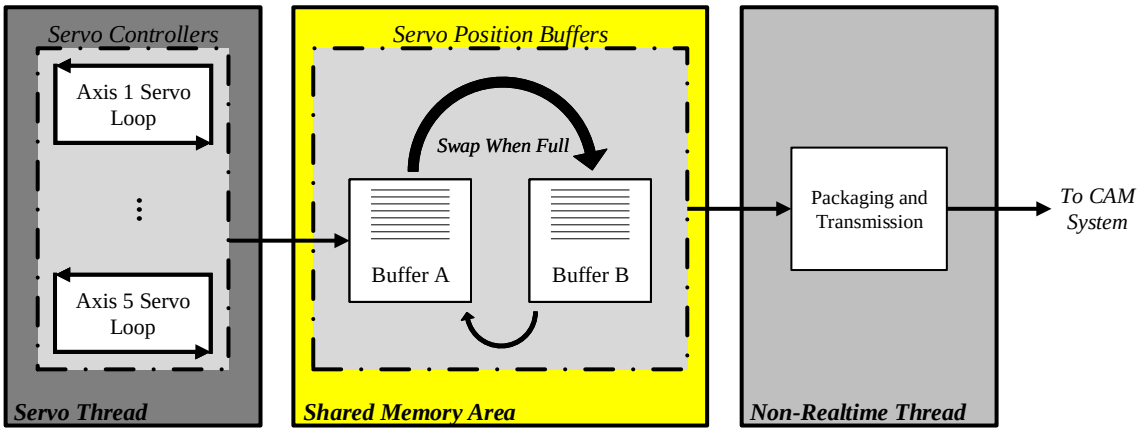


Figure 3.2: Alternating Buffer Configuration

emptied buffer is available for storage of current servo position information.

3.2.2 Non-Realtime Components

Servo commands and feedback information are moved into and out of the shared memory area by the task executor process. The task executor is responsible for commanding the realtime process and updating user-accessible state machine information. Interaction with the task executor is performed by a non-realtime process written in C++ that is running in user-space. This process, referred to as the Machinekit Remote Shell (MKRSh) contains three separate threads that are responsible for communication with the CAM system and servicing control and feedback tasks, respectively. MKRSh is primarily a transmission control protocol (TCP) server that accepts connections from remote clients (e.g., a CAM system) for the purpose of controlling the realtime subsystem of the machine tool. MKRSh has access to the shared memory area used by the servo thread, and can also access the full network stack for communication with the outside world.

Network Data Format

Two command formats for communication between MKRSh and a remote client were developed for this dissertation. The first format is a simple human-readable text-based (Uni-

code) format, where commands have an intuitive name. Additionally, the command is preceded by either the ”GET” or ”SET” word to signal to MKRSh which direction data is to flow upon processing of the command. For example, to set the buffer size for servo position data, the command syntax is:

SET SERVO_LOG_PARAMS Arg1 Arg2 Arg3 Arg4 Arg5 Arg6

The SET word denotes that this is a command to set the value of data in the realtime system; SERVO_LOG_PARAMS is the data item to set (the servo logging parameters settings structure), which serves as the message header; Arg1 is a flag (0 or 1) to disable or enable recording of servo feedback information; Arg2 is an integer that denotes the number of axes to log; Arg3 is an integer to denote the subsampling rate to use; Arg4 is an integer to denote the size of the alternating feedback buffers; Arg5 is a flag to denote which of the two feedback buffers (A or B from Figure 3.2) to write to; and Arg6 is a flag to denote that the buffers should be initialized as empty arrays. The space character is used as the delimiter between consecutive items in the message.

The Unicode command format works well for low frequency commands, like setting the desired machine state (e.g., emergency stop, drive power, servo logging parameters, etc.), but is inefficient for transmission of high frequency numeric data that is used for actual motion control. For motion control purposes, a binary format was developed that uses fewer bytes to transmit the same information than would be possible with Unicode. The binary format transmits numeric data as 32-bit floating point numbers in the IEEE-754 floating point format [119]. This binary format is used for both command and feedback data during actual machine operation, and the Unicode format is used only for preparatory commands. The binary format consists of three elements: two header bytes, which denote the type of data in the command packet; four bytes representing an unsigned integer that convey the total number of bytes that will be in the feedback packet; a variable size payload that consists of 32-bit floating point numbers, where the length of the payload in bytes is an integer multiple of the number of motion axes of the machine; and a delimiter byte which

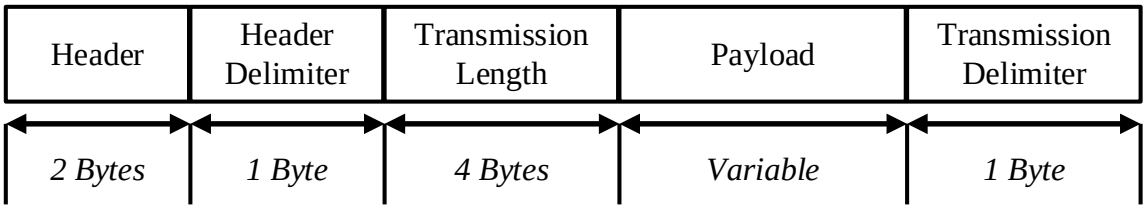


Figure 3.3: Binary Data Format

signifies the end of the transmission. Figure 3.3 illustrates the construction of a packet in the binary transmission format. The delimiter byte was chosen as the byte that signifies not-a-number (NaN) exponent in the IEEE-754 format, which is 0x7f. Under normal operation, this byte will not appear in the payload of motion data, which enables a client to detect the byte as a delimiter.

Connection Launch Thread

The main thread of MKRSh runs a TCP server that listens for client connections and spawns independent control and feedback threads for each connection. This thread runs continuously to respond to connection attempts from the CAM system. MKRSh can service multiple simultaneous CAM clients to enable data collection on different systems at the same time.

Control Thread

The control thread of MKRSh is responsible for transferring servo position commands from the TCP socket to the realtime process that runs the servo controllers. The thread runs a main loop that checks for available bytes on the TCP socket and assembles them into complete transmissions. Construction of transmissions for command data follows the binary format described in Section 3.2.2. The number of commands in a complete transmission must be an integer multiple of the number of controlled axes of the machine tool, though the number of time samples is controllable by the user. In other words, the user can transmit any number of complete servo samples, where each complete servo sample has five

entries. The control thread simply waits for the receipt of the end delimiter byte to determine when the transmission is complete. The functionality for variable size transmissions enables reliable data exchange performance for network configurations with unpredictable latency.

Upon receipt of a complete transmission, the control thread assembles an NML message that contains the received servo commands as an array with appropriate metadata to signal that the NML message is to be interpreted as a direct control command. The message is then sent to the task executor to be placed in the shared memory area where it is picked up by the servo thread. Once the servo thread picks up the command array, it places each position sample into the trajectory buffer.

Feedback Thread

The feedback thread is responsible for transferring recorded servo positions to the remote client and for informing the client of the current trajectory buffer fill level so the client can modulate transmission frequency accordingly. Because the CAM system is running on a non-realtime platform, the trajectory buffer enables absorption of non-deterministic latency between the two systems and allows the CAM system to provide high-frequency motion commands to the realtime system without interrupting motion execution. Reading of the current buffer fill level across the RT/non-RT boundary is essential to effective control of the buffer fill level. Additionally, the feedback thread notifies the task executor when the current sample of position feedback information has been written to the TCP socket, so that the realtime process can be informed that one of the alternating buffers in 3.2 is ready to be filled again. The software architecture of the MKRSh process and its communication with the task executor and realtime process is shown in Figure 3.4.

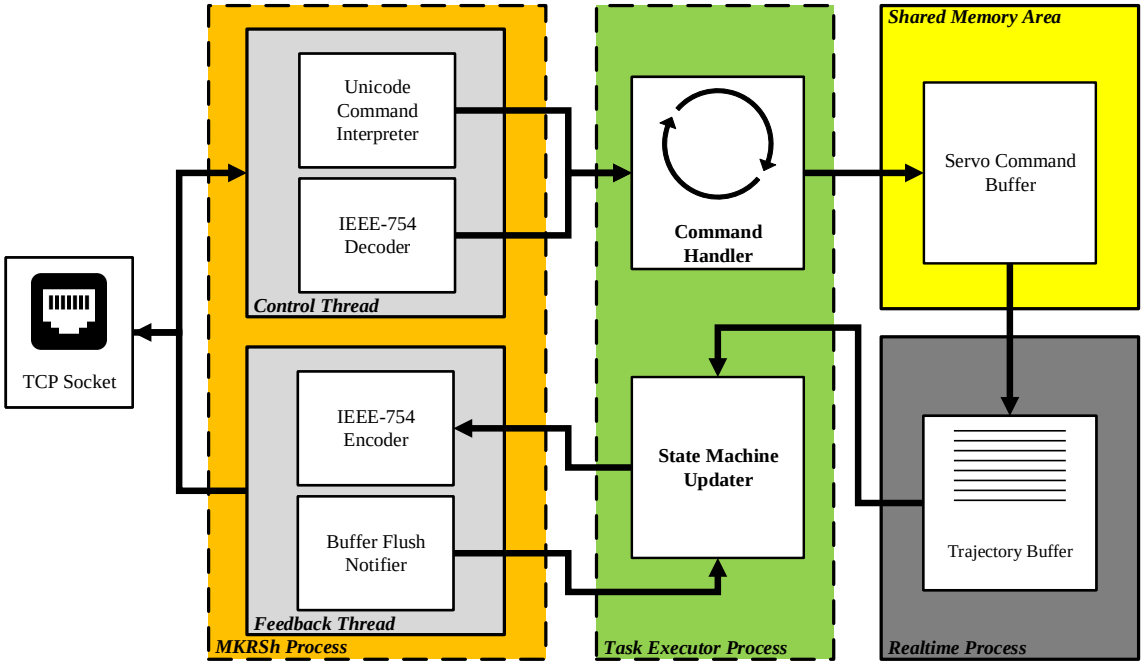


Figure 3.4: MKRSh Software Architecture

3.3 CAM-CNC Interface Application

High speed data transmission between the realtime control system on the BBB and the voxel-based CAM system necessitated the development of interface software to manage communication and data collection. This interface layer was written in Python and configured to run entirely on the central control computer. The interface application is responsible for the following four broad tasks:

1. Generation and transmission of servo position and preparatory commands to the realtime control system

2. Processing of feedback information from the realtime control system

3. Communication with the external data acquisition system used to measure encoder position

4. Storage and management of data collected during machining

5. Communication with external clients, such as the CAM system or a terminal

6. Management of shared states within the interface application

The Python application was developed using Python's multiprocessing library [120], which enables process-level parallelism by launching individual python interpreters for each process. Each of the broad tasks described above was assigned to a separate process for high-performance concurrent execution. Because Python is an interpreted (as opposed to compiled) programming language, a Python interpreter must translate user-written code into machine instructions during execution; to maintain thread safety, the developers of the Python language implemented what is known as the Global Interpreter Lock (GIL) which ensures that only one thread can execute user code at a given time [121]. While use of the GIL helps to ensure thread safety, it can also degrade performance in highly parallel applications. Because each of the four tasks outlined above must execute simultaneously during machine operation, a multiprocess architecture was chosen to bypass the GIL. Thus, each of the major tasks, many of which have multiple simultaneous *threads*, were written into individual processes, each with its own Python interpreter (and correspondingly, its own GIL). One master process (item 6 in the above enumeration) was designed as the manager of the other processes, and is responsible for handling shared states and handles to relevant objects used by the other processes. Additionally, the master process handles communication with external clients. A diagram of the software architecture of the interface application is shown in Figure 3.5. The heavily dashed arrows denote the direction of shared state data flow between processes, the lightly dashed arrows denote machining data flow, and red arrows denote interprocess control command flow. This section describes the architecture and operation of each of the individual processes.

3.3.1 Control Process

The control process is at the heart of the interface layer, as its primary responsibility is to send the necessary commands for the machine to run a given toolpath. The control process

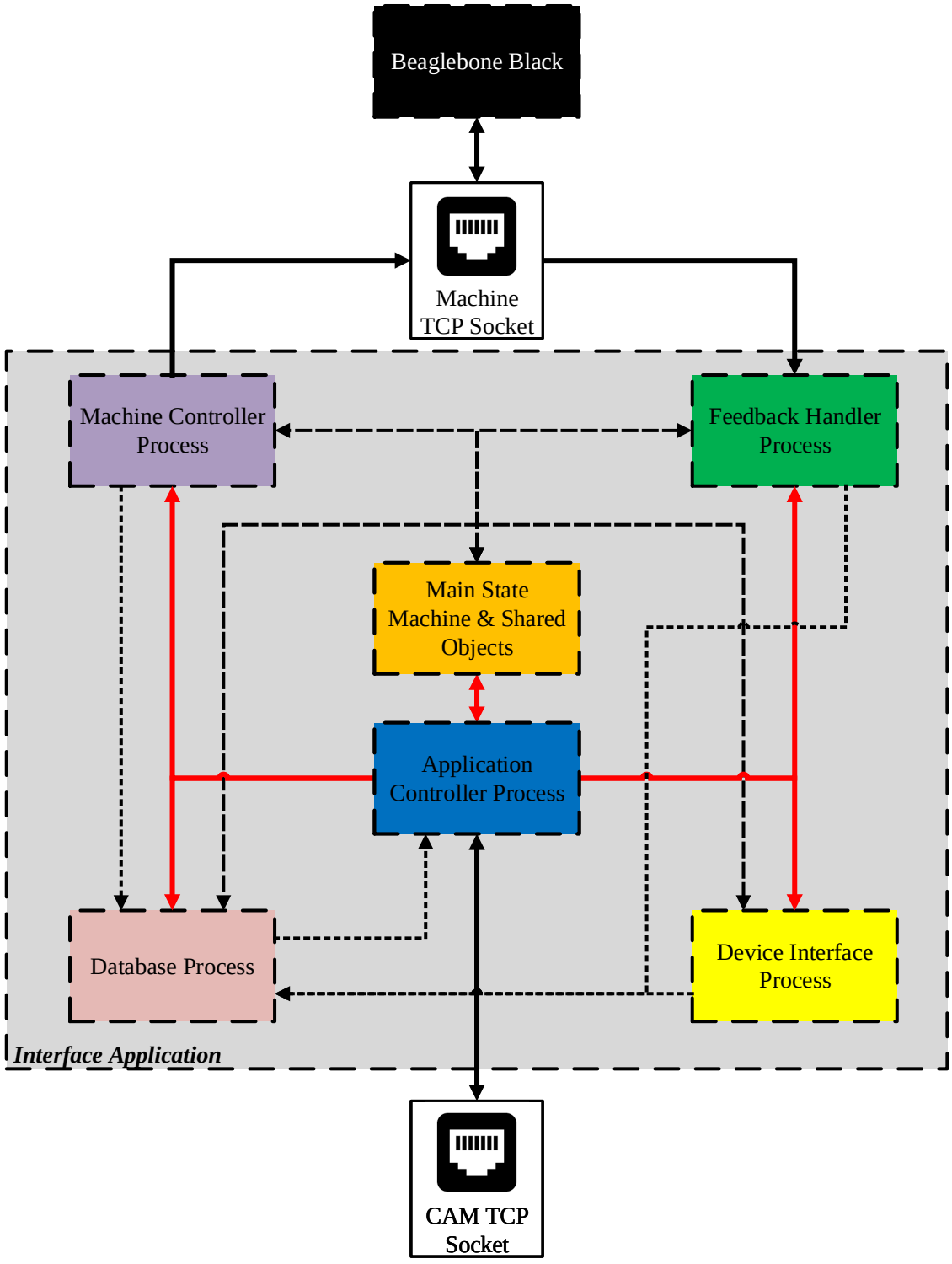


Figure 3.5: Interface Application Architecture

consists of three components, each of which runs on separate groups of threads within the process:

1. Controller Manager: One thread responsible for executing commands sent to the control process by other processes in the interface application, such as a connection or execute request, and another thread responsible for maintaining a secure shell (SSH) connection to the operating system of the realtime control system

2. Motion Controller: Two threads responsible for high frequency transmission of servo points to the realtime control system

3. Websocket Interface: Two threads responsible for managing a websocket used to communicate with a remote trajectory planning system

Controller Manager

The main thread of the controller manager runs the command handler that executes machine commands sent by the CAM system or a debugging terminal. These commands are used to initialize communication and prepare the machine for motion. The possible commands that the manager can execute are:

- CONNECT: Initializes an SSH connection to the BBB OS and starts MKRSh, initializes a socket connection to MKRSh, checks the machine home state, applies power to axis motors, synchronizes state machines maintained in the interface application and synchronizes clocks

- PLAN: Creates a trajectory plan from raw voxel points obtained by the CAM system by using the websocket interface to the remote trajectory planner

- EXECUTE: Starts the motion control threads, ensures machine is in the correct state to start cutting, generates a rapid reposition from the current position to the cutting

start position, and begins commanding high-frequency servo points to drive the machine along the toolpath

Motion Controller

The motion controller is one of the most important components of the interface application, as it is what enables the machine to drive the axes through the toolpath. The motion controller component consists of two separate threads: a motion control communication thread that writes servo commands to the TCP socket connected to the machine at a controlled rate, and a queue feeder thread that enqueues constant-size motion blocks for the motion control thread to consume. The queue feeder thread the precursor to motion generation. The queue feeder obtains complete moves from the trajectory planner, and then segments them into constant size blocks and places them in a queue. This is done asynchronously from the actual motion command communication, since variable size motion blocks can cause unpredictable latency in queue retrieval for the motion communication thread. Once the constant size motion blocks have been created and enqueued, the blocks are available for execution by the motion communication thread. The size of the motion blocks is a user parameter. Both the motion queue feeder and the motion control communication thread make use of various synchronization primitives (primarily thread events) that are shared among all threads and processes in the application. These primitives are written to or notified by each thread as needed. A diagram of the motion queue feeder thread is shown in Figure 3.6. Both the motion queue feeder and the motion control communication thread are started upon receipt of a start signal from the main machine controller thread.

The motion control communication thread retrieves constant size motion blocks from the motion block queue and then further segments them into packets (using a packetizer) that are suitable for single-shot transmission on the TCP socket to the machine tool. Each packet is formed in the same network data format shown in Figure 3.3, where the payload has an integer multiple of the number of axes of the machine tool under control. Each servo

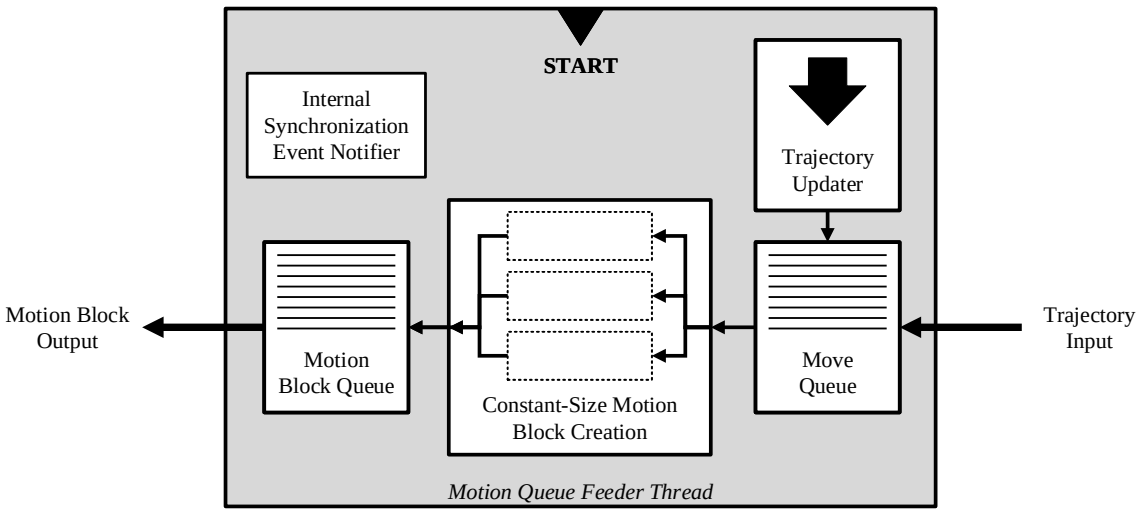


Figure 3.6: Motion Queue Feeder Thread

command is again a 32-bit floating point number in the IEEE-754 format. The number of samples in the payload, where each sample corresponds to one servo period, is controllable parameter that can be adjusted to account for variable network latency and CPU power of the main control computer. Additionally, the number of NML messages that are constructed from the payload is controllable by the user if there is a desire to send more samples in a single transmission than are transferrable in a single NML message.

The motion control communication thread operates a proportional controller to control the rate of data transmission in response to the current level of the buffer. The proportional controller can directly control the number of servo points in the trajectory buffer, which corresponds to some level of temporal ”lookahead”; for example, if the current buffer level is 500 points, the machine have 500 servo periods worth of motion data that can be executed until motion is halted. The buffer level controller ensures that the machine does not run out of motion data, even in the presence of network uncertainty. For this application, a proportional controller was sufficient since the absolute fill level of the buffer is not of primary importance; rather, the goal is simply to ensure that the buffer level is neither completely empty nor completely full. Steady state error in the actual fill level is tolerable. A pictorial depiction of the buffer level controller is shown in Figure 3.7. The controller

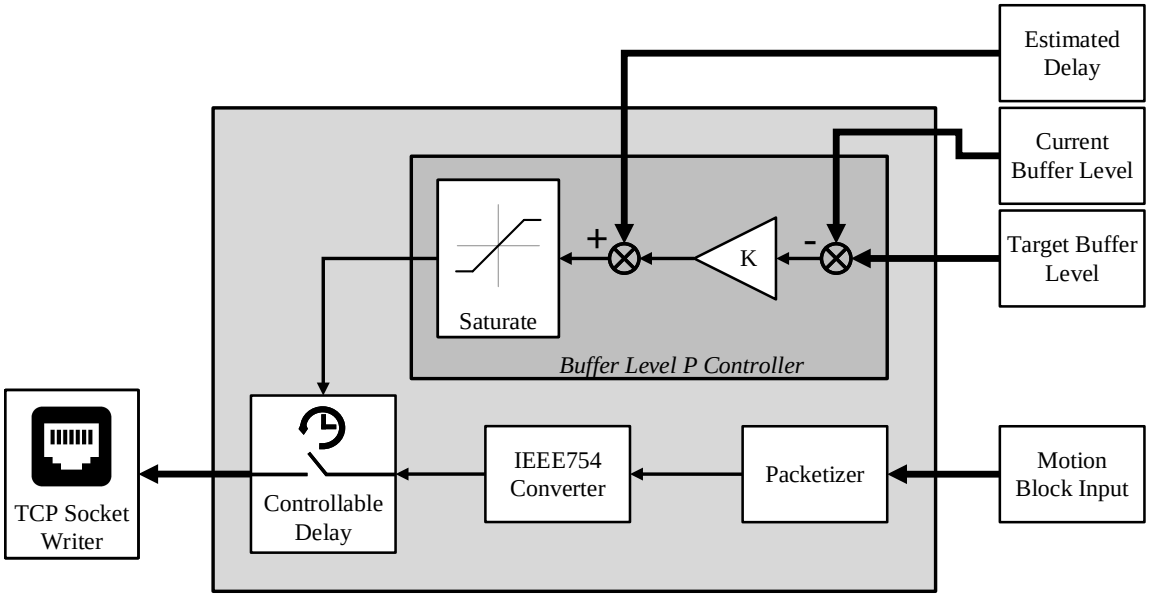


Figure 3.7: Proportional Controller for Buffer Fill Level

operates according to Equation 3.2,

$$D = T_{\text{Servo}} \max(0, N_{\text{Packet}} - K(L_{\text{Target}} - L_{\text{Current}})) \tag{3.2}$$

where D is the time delay, T_{Servo} is the servo period, N_{Packet} is the number of servo samples in the last packet that was sent (which corresponds to the estimated delay in Figure 3.7), L_{Target} is the target buffer fill level, and L_{Current} is the current buffer level when control action is performed. The controllable delay is implemented as a timed wait in code, and introduces a delay between transmission of consecutive point packets. The proportional gain K is tunable by the user to account for differences in network configurations. The saturation block is used to prevent a negative delay from being commanded. All motion commands that are sent on the TCP socket are thread safe, meaning that access to the socket is protected by a mutex to prevent interference between motion commands and any other commands that the main thread of the machine controller process may need to send out during motion execution (such as disabling drive power or emergency stopping the machine).

Trajectory Planner Interface

The actual motion trajectories are generated by a remote trajectory planning (TP) server that is supplied with the raw toolpath position samples from the voxel model. The control process communicates with the remote trajectory planning server using a websocket through the publicly-accessible and free-to-use Achex gateway, which enables peer-to-peer communication of two disparately located machines through the Internet. The Achex gateway serves as a middle man for socket communication: both the main control computer and the remote trajectory planning server connect to the gateway, and packets sent to the gateway by one participant are forwarded to the other. As a result, the trajectory planning server needs only an Internet connection and does not need to be co-located with the main control computer.

The raw voxel point samples are provided to the interface application by SculptPrint, where they are parsed and stored into a local array. The raw voxel points are subdivided by movement type, meaning that reposition and reorientation moves which do not involve cutting of material are coded as rapid moves, and feed moves that do involve cutting of material are coded as cutting moves. Each move is assigned what is known as a sequence number, attached to a unique instance of the Move class, and enqueued into a FIFO (known as the ”voxelized path queue”) to be sent to the trajectory planner. This operation is completely asynchronous from the execution of the motion controller component, and planned trajectories can be executed at any time. The trajectory planner interface component is constantly connected to the remote trajectory planning server while the interface application is running. Upon receipt of a move object from the voxelized path queue, a websocket writer thread packages the move points with any necessary metadata, serializes and encodes the resulting object in a text-based format suitable for web communication known as Base64, and dumps it to the websocket. Meanwhile, a separate websocket reader thread constantly listens on the websocket for receipt of either planned trajectories or control messages from the trajectory planning server. Trajectories are received in constant-size chunks

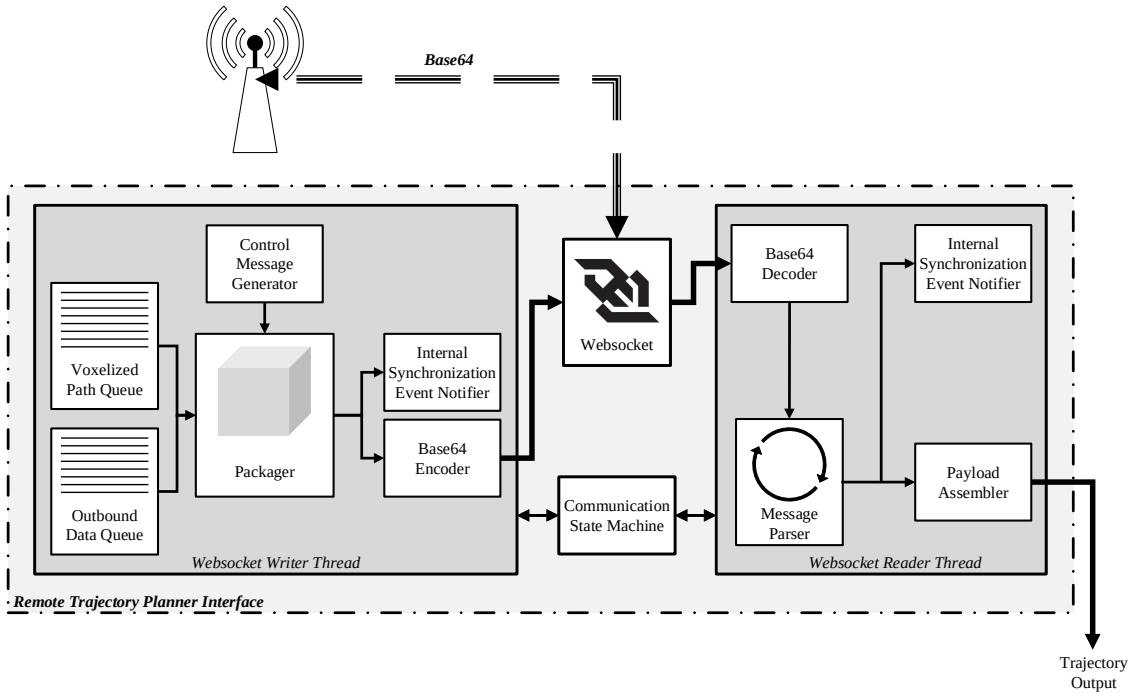


Figure 3.8: Websocket Interface

to respect transmission size limitations imposed by the Achex gateway. Control messages are handled by a message parser, and planned trajectory chunks are assembled by a payload assembler and then forwarded to the motion queue feeder thread. An internal state machine is maintained by the trajectory planner interface component, which keeps track of which trajectories are out for planning and which have been received. A diagram of the websocket interface is shown in Figure 3.8. The complete control process diagram is shown in Figure 3.9.

3.3.2 Feedback Handler Process

The feedback handler process is responsible for collecting, parsing, and handling all bytes that are received on the TCP socket that is connected to the realtime controller. The feedback handler continuously listens on the TCP socket for available bytes, which are formatted according to Figure 3.2.2. During normal machine operation, low-frequency control messages are written to the socket from the realtime controller in the Unicode format, and

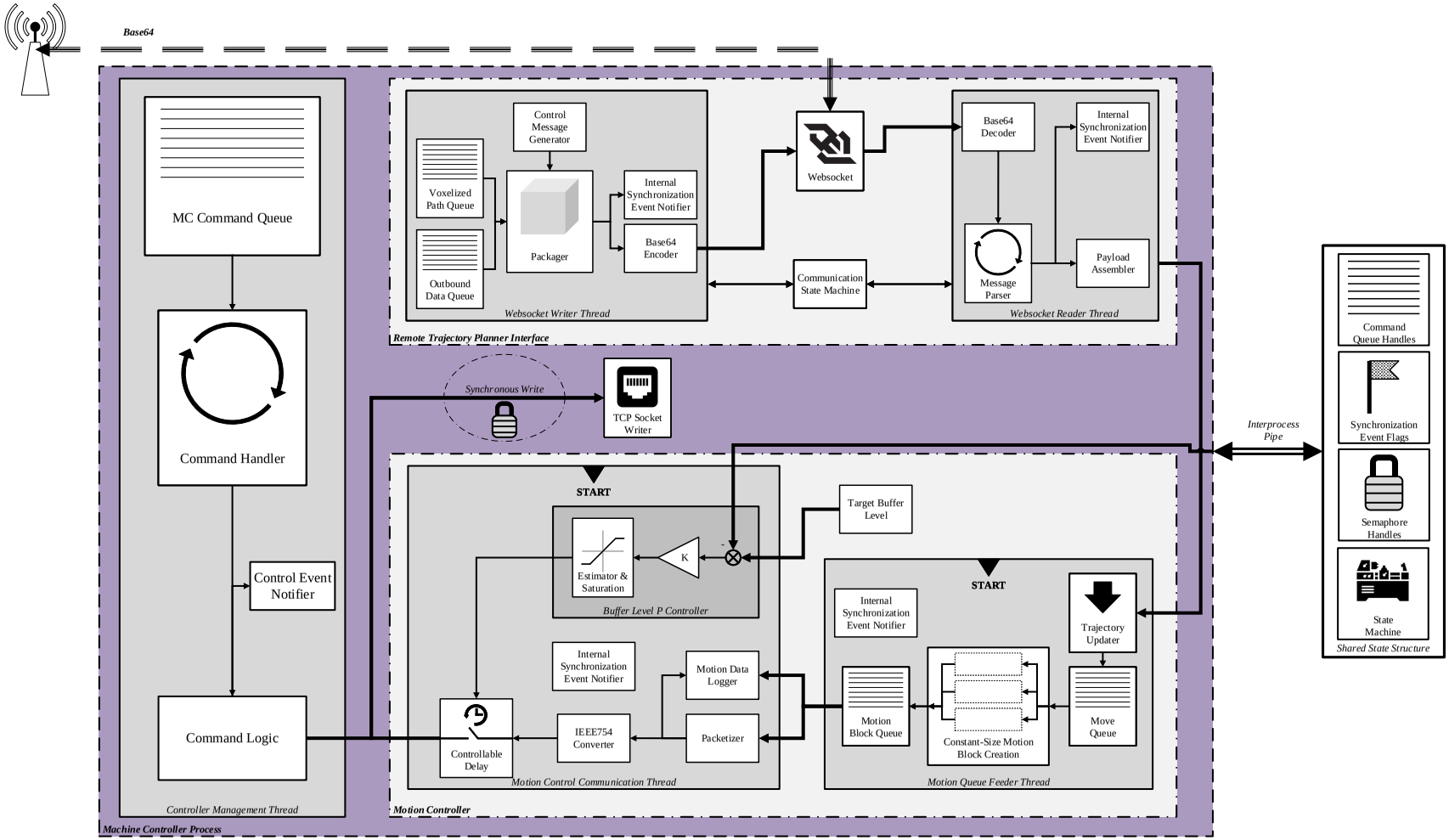


Figure 3.9: Control Process

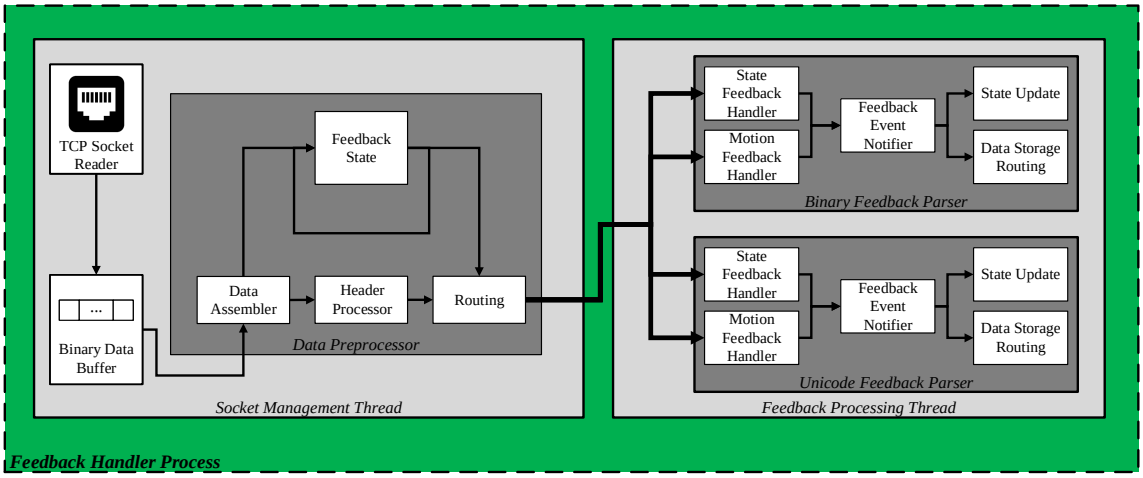


Figure 3.10: Feedback Handler Process

high-frequency motion data are written in the binary format. The feedback handler process contains two separate threads, one of which buffers bytes that are received on the socket and packages them into complete transmissions, and one that processes those transmissions, stores the relevant data from them, and updates the main state machine accordingly. A diagram of the feedback handler process in shown in Figure 3.10.

The socket management thread collects received bytes from the TCP socket, stores them in a data buffer, and logs all bytes that are received on each socket read into the database described in Section 3.3.4. The socket read operations are performed whenever the OS of the main control computer signals to the application that bytes are available on the socket, and the actual number of bytes may or may not correspond to a complete transmission from the realtime controller. As a result, the socket management thread may need to take multiple passes on the socket to receive all of the bytes for a single transmission, as delimited by the 0x7f delimiter byte. The thread maintains an internal state machine that stores information about receipt of transmission headers, delimiters, and payload sizes.

The feedback processing thread contains two logic paths: one for parsing feedback packets that are constructed in the Unicode format, and one for parsing packets that are constructed in the binary format. Because the feedback processing thread runs asynchronously from the socket management thread, it can process feedback messages while the socket

management thread is assembling the next message. This enables higher parallelism in operation of the interface application. The feedback processing thread routes complete feedback message to the appropriate logic path depending on the header bytes in the message. Upon successful parsing of the message, thread synchronization events are notified, the main state machine is updated accordingly, and the data contained in the message are routed to the appropriate data storage memory location.

Synchronization of Main State Machine

Commands that are issued to the realtime system must be acknowledged and confirmed by the BBB before the main state machine is updated in the interface application. Additionally, because commands are issued from one process (the main thread of the machine controller process) and received by another (the feedback processing thread of the feedback handler process), synchronization objects must be used to ensure that the main state machine is updated at the correct time.

3.3.3 Device Interface Process

The device interface process handles all communication with external hardware devices that are relevant to machine operation; in the current iteration of the experimental machine, the two external devices are the quadrature decoder and the spindle drive. Each device is different and communication is asynchronous, so a separate thread was created within the process to communicate with each device. The device interface process serves as a slave process to the machine controller process, and all of the actions of the device interface are performed in response to requests from the machine controller.

The spindle drive interface thread runs a command handler on an endless loop that processes commands that are enqueued in a command FIFO, such as spindle speed and position commands. The spindle drive communicates with the interface application using a Universal Serial Bus (USB) connection and exposes a state object to the interface applica-

tion that can be written to or read from to set or get various drive parameters. At machine startup, the interface application establishes communication with the spindle drive, sets the necessary position and velocity loop gains of the drive, and awaits a command from the machine controller.

The encoder interface thread is somewhat more complicated than the spindle drive interface thread becuase it does not rely on the exposure of a remote object from the device for communication purposes. Instead, the encoder interface thread communicates with the quadrature decoder board using 5V UART connection. Upon startup, the encoder interface thread bootstraps the quadrature decoder board, sets the baudrate for communication, and synchronizes the encoder counts to the current machine position provided by MKRSh. After initialization is complete, the thread continuously requests the current encoder count from the quadrature decoder, buffers a user-selectable number of encoder readings, and pushes full buffers of encoder readings to the database. Each encoder reading is clocked upon receipt to enable correlation with commanded servo points that are sent to the machine by the motion controller. Encoder readings are sent by the quadrature decoder as 32-bit unsigned integers, which are converted to floating point numbers using appropriate scaling factors for each motion axis. The encoder data format is similar to the commanded data format shown in Figure 3.3.

3.3.4 Database Process

The database process handles and stores all data that are collected by the other processes of the application. The database must be able to quickly transfer data into and out of memory in response to requests by the other processes in the interface application, and also must be able to store the large amounts of data that are collected during machine operation. There are a total of four individual threads in the database process:

1. Command Handler Thread: Accepts requests by other processes to store or retrieve data from the database, writes the entire database to the hard disk or an external

client, and updates the main state machine

2. Puller Thread: Synchronously performs data retrieval from the database when given one or more data tags and the corresponding sample index ranges to retrieve

3. Pusher Thread: Asynchronously writes data to the database, when provided with one or more data tags and data payloads for each tag

4. Disk Logging Thread: Writes archival data (which will not need to be retrieved during machine operation) to a file handle on the hard disk

The command handler thread picks up database commands from a command queue and processes them one-by-one. The commands that have been implemented are: pull, push, flush to file, flush to websocket, and log to disk. Received commands are parsed and routed to the appropriate thread by the command handler. Upon completion of routing for a command, the command handler updates the main state machine by pulling the latest machine position from the database. Each database command is a unique object of the DatabaseCommand class.

The puller thread retrieves data from the database in a synchronous fashion, meaning the entire database memory area is locked during execution of a pull command. Locking is performed with a mutex, and no push commands are processed while a pull command is being executed to protect data integrity. Additionally, to preserve ordering of pull commands, any process that requests a pull from the database waits until the puller thread has returned the appropriate data before continuing their execution. The puller thread can pull multiple data tags with different index ranges in a single command to both limit the number of times the database mutex must be acquired and to ensure that all data gotten during a pull have the same end time.

The pusher thread waits until the receipt of a push command from the command handler thread and then performs insertion and commit to the database. The pusher thread, like the puller thread, locks the database memory area to prevent simultaneous access by the puller

and pusher threads. The pusher thread can also dynamically modify the database to add data tags that are pushed in if they do not already exist in the database. To limit the number of times the database mutex has to be locked, the pusher thread can process an arbitrary number of data tags in a single commit.

The disk logging thread is used to record debugging information to a file handle on the hard disk of the main control computer. All of the data that are received on the TCP socket by the feedback handler are pushed as log commands to the database command handler, which forwards them to the logging thread. The logging thread picks up a log command from a FIFO at every iteration and writes it as text to a log file handle. The file handle is flushed on every write. This functionality is useful for debugging any issues encountered with feedback data from the realtime control system.

The actual database is stored as an object in memory with a variable number of arrays that are initialized either at startup or dynamically by the pusher thread. The arrays utilize the NumPy library for performance and functionality. The database also contains lists to store objects that are created during machine operation, such as movement commands that are obtained from the trajectory planner. During a database insertion and commit, the appropriate data arrays are appended to and stored; during a database pull, the appropriate arrays are indexed and the relevant entries are copied out of them and into a list that is returned to the requester. The data items and their corresponding types that are used during machine operation are shown in Table 3.1, where the data types denote a single time sample (in the case of numeric data) or a single object (in the case of object storage). A diagram of the database process is shown in Figure 3.11.

Source	Data Item Name	Data Type
Xenomai	Realtime Clock	32-bit Float
	Stepgenerator Position	1x5 Vector of 32-bit Floats
MKRSh	Linux Kernel Clock	32-bit Float
	Buffer Level	Unsigned Integer
Quadrature Decoder	Serial Data Receipt Time	32-bit Float
	Encoder Position	1x5 Vector of 32-bit Floats
Feedback Handler	Stepgenerator Feedback Receipt Time	32-bit Float
	Buffer Level Feedback Receipt Time	32-bit Float
Motion Controller	Transmission Send Time	32-bit Float
	Commanded Servo Point Transmission	Object

Table 3.1: Data Items in Database During Normal Operation

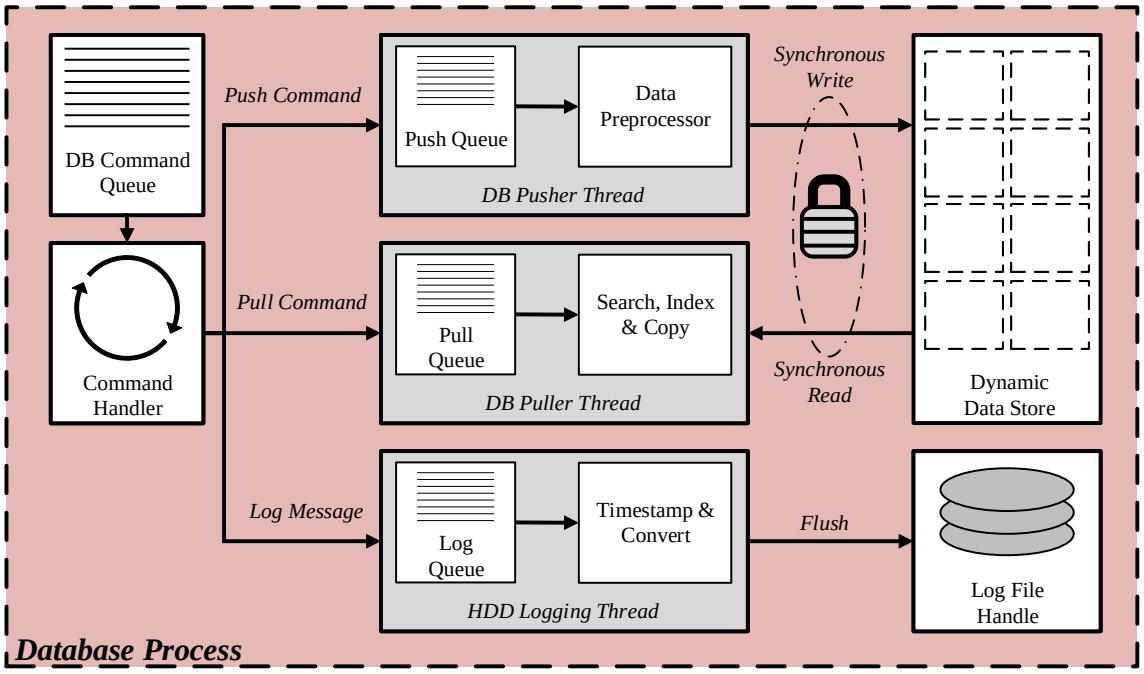


Figure 3.11: Database Process

3.3.5 Application Controller Process

The application controller process is the master component of the interface application that is responsible both for administration of worker processes and communication with external clients, such as the CAM system or a debugging terminal. The application controller was designed to accept any number of CAM systems or debugging terminals to be connected simultaneously to enable remote operation of the machine tool. The application interface process has two main threads that run continuously, and a variable number of threads that are launched depending on client connections. The main application controller thread (appController) is the master executor of the entire interface application, and is responsible for handling all commands from external clients and routing them to the appropriate worker processes. The socket launcher thread is responsible for listening for external TCP socket connections to the main control computer from an external client and then launching separate socket communication threads for each individual connection. The socket communication threads listen for data on the established TCP connection and forward commands or feedback information between the socket and the appController thread.

The appController thread follows a similar architecture to the main threads of the other processes, in that it continuously runs a command handler to interpret and process commands that are passed in by an external client through a socket communication thread. The appController is also responsible for the launch and initial synchronization of the worker processes, which is not a trivial task given the relative complexity of the interface application. The processes and their corresponding threads must be launched in a particular order to ensure that the application starts correctly. The appController surrenders control of the machine tool to the external client interface upon successful startup of the worker processes. The currently implemented commands that are usable by an external client are:

1. INIT: Instantiate shared objects, start SSH connection to realtime control OS, and determine execution state of MKRSh on BBB

2. START: Launch worker processes and initiate startup sequence for their corresponding threads

3. CONNECT: Forwarded to the CONNECT method of the machine controller process once the startup sequence is complete

4. PLAN: Forwarded to the PLAN method of the machine controller process once the startup sequence is complete and the remote trajectory planner handshake has been completed

5. EXECUTE: Forwarded to the EXECUTE method of the machine controller process once the motion control threads have been started and the websocket interface has indicated that usable trajectories have been returned by the remote trajectory planning server

6. FASTFORWARD: Initializes the current clock time as the start time for reading feedback data

7. READ: Pulls all motion data from the database that has been acquired since execution of the last READ request and sends the data to the external client

Two methods exist for sending commands from an external client to the appController: an OS pipe (known as a named pipe in this Windows implementation, though a UNIX socket could also be used if the main control computer was running a UNIX-like OS) or a standard network socket. The network socket approach is preferred as it is more portable between operating systems of the main control computer. In the present implementation, where the CAM system and the interface application are run on the same main control computer, a local TCP loopback socket was used. However, it is equally possible to use a standard TCP network socket to connect an external CAM system to the network interface of the main control computer (i.e., the CAM system would be run on a separate machine). Commands are typically sent as serialized command objects, but they can also be sent as Unicode text,

which would be used by a local or remote debugging terminal connected to the outward-facing socket of the appController.

3.3.6 Interprocess Communication and Shared Object Management

Communication between processes in the interface application is accomplished using a manager process, whose constructor is provided by the Python multiprocessing library. The manager is responsible for two main functions: maintenance and manipulation of the main state machine, which inherently necessitates a thread-safe implementation; and construction and provisioning of proxies to shared objects that cannot be serialized for transmission between other processes (unserializable objects would lose context if they were serialized and sent between processes). Such unserializable objects include semaphores, queues, and pipes. The proxy objects are used by the worker processes to indirectly affect the proxies' referents that are stored in the manager process.

The majority of interprocess communication takes place using queues, such as the command queues that are used by many of the worker processes. The command queues are FIFOs that are appended to through a queue proxy that is provided by the manager, and queue entries are consumed by the process to which the queue belongs. For example, if the application controller were commanded by the CAM system to initiate the TCP connection to the realtime system, the application controller would place a CONNECT command on the queue of the maachine controller process. Likewise, if the machine controller needed to extract data from the database (to be flushed to the remote trajectory planning server, for example), the machine controller would place a PULL command on the database command queue and wait for data to be returned by the database process. Because the command queues themselves reside in the manager process and proxies to each queue are provided to each worker process, any worker process can command any of its colleague processes. However, some interprocess communication is accomplished using simple duplex pipes between processes. For instance, on application startup, object proxies are send through

pipes from the manager to each worker process.

Synchronization primitives, such as mutexes and events, are also provided by the manager. These primitives are used to ensure thread safety and the correct order of execution for certain tasks. Take, for example, the case of the machine controller and feedback handler processes: the machine controller is responsible for setting the state of MKRSh to ready the machine for certain operations, but all feedback information on the *current* state of MKRSh is provided to the feedback handler process. As a result, the machine controller process must wait on the feedback handler to acknowledge that MKRSh has performed a state change before modifying the main state machine of the interface application. Synchronization of state is performed in the following way:

1. The main thread of the machine controller process notifies a proxy to a mutex that protects a state change event flag that a state change is requested and begins blocking. Meanwhile, the feedback handler process receives and decodes feedback bytes from the TCP socket to MKRSh as normal.

2. The manager process receives this notification through a pipe to the proxy, and locks the proxy's referent (the mutex itself)

3. The manager process notifies the machine controller that the mutex has been acquired

4. The main thread of the machine controller process then continues execution, clears the state change event flag, notifies the proxy to the state change flag mutex that it can unlock, and begins blocking

5. The manager process receives notification through the pipe to the proxy for the mutex that protects the state change flag that the mutex can be unlocked; the manager unlocks the mutex and informs the requester (the main thread of the machine controller process) through the proxy that the mutex has been unlocked

6. The main thread of the machine controller process continues execution and locks another mutex protecting the TCP socket to the machine through a separate proxy

7. Once the TCP socket mutex has been acquired, the main thread of the machine controller process commands MKRSh to change state and begins continuously polling MKRSh for the current state on a timed loop

8. The feedback handler process continues processing feedback data until it receives notification from MKRSh that the requested state change has been completed. Upon acknowledgement of the state change, the feedback handler requests the manager to lock the mutex that protects the state change flag through the proxy to the semaphore

9. Once the mutex has been acquired, the feedback handler writes the state change flag and unlocks the mutex through the proxy

10. The machine controller is notified by the manager that the state change flag has been set, at which point it terminates polling of current state from MKRSh

11. The machine controller requests a lock of a separate mutex that protects the main state machine itself (maintained in the interface application) through a separate proxy

12. The manager process notifies the machine controller through the mutex proxy that the mutex has been locked

13. The machine controller writes the state change to the main state machine in the interface application, and requests an unlock of the mutex that protects the state machine

14. The manager unlocks the state machine mutex and notifies the machine controller process that the mutex has been released

15. The machine controller requests an unlock of the mutex that protects the TCP socket to MKRSh

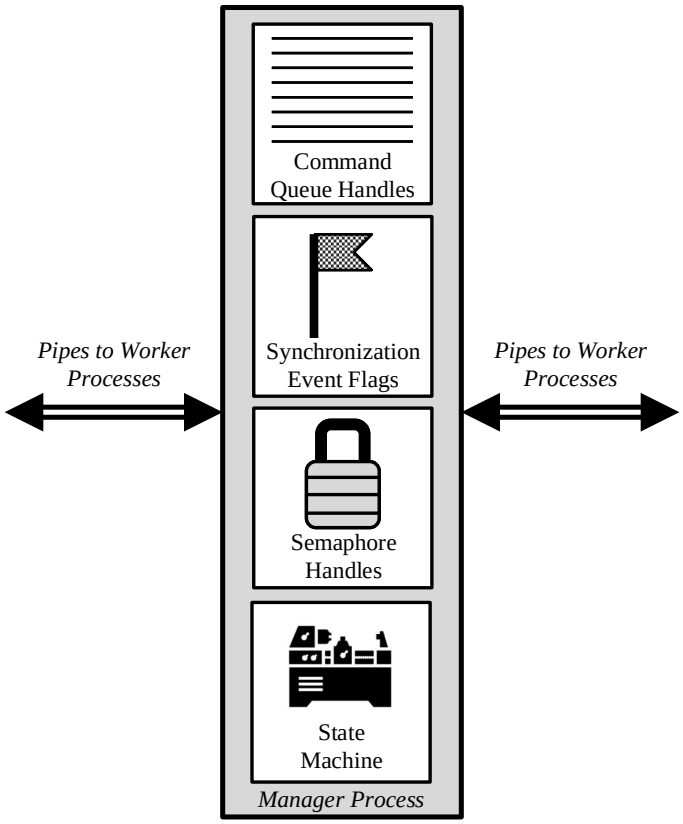


Figure 3.12: Manager Process

16. The manager process unlocks the semaphore that protects the TCP socket and notifies the machine controller process

This completes the state change procedure, which is repeated for any state change that is commanded of MKRSh by the machine controller process. A similar procedure is followed for state changes that are initiated by MKRSh (such as an emergency condition, about which the machine controller must be notified immediately). This functionality has been neatly packaged into the manager process, which encapsulates much of the low-level synchronization into functions written for the interface application. A diagram of the manager process is shown in Figure 3.12.

3.3.7 Process Summary

The entirety of the interface application relies heavily on parallel processing and concurrent programming techniques. Many different tasks need to be managed simultaneously to enable effective control of the machine tool, which required implementation of many parallel code paths during development. The interface application consists of approximately 5000 lines of Python code. Table 3.2 summarizes all of the individual processes and their respective threads that were implemented for the interface application.

3.4 Integration with SculptPrint

The SculptPrint CAM system functions as a normal TCP client from the perspective of the application controller, and can execute commands like any other client. SculptPrint is equipped with a Python API that enables integration of user-written code that communicates with the application controller. Because SculptPrint runs on the same physical machine as the interface application, it communicates with the application on a loopback socket. Various user controls and graphics windows in SculptPrint enable the control of the machine tool and visualization of command and feedback data. An image of the SculptPrint interface that was developed to support the machine control application is shown in Figure 3.13, which depicts realtime visualization of the location of the cutting tool on the workpiece, in addition to two dimensional axis velocity (in the bottom left) and trajectory buffer performance (left top and left middle plots).

Process Name	Thread Name	Purpose
Machine Controller	Command Handler	Receives and routes commands from other processes
	OS Controller	Sends OS commands over SSH to BBB
	Motion Controller	Sends points over TCP to MKRSh
	Motion Queue Feeder	Creates constant-size motion blocks for motion control thread
	Websocket Reader	Receives trajectories TP server over the websocket
	Websocket Writer	Requests trajectories from TP server over the websocket
Feedback Handler	Socket Manager	Receives and assembles bytes from TCP connection to MKRSh
	Feedback Processor	Processes complete feedback messages assembled by the socket manager
Database	Command Handler	Receives and routes database commands
	Pusher	Inserts into and commits database
	Puller	Retrieves data from database
	Disk Logger	Writes log data to disk
Device Interface	Spindle Controller	Communicates with spindle drive over USB
	Quadrature Board Controller	Communicates with quadrature decoder board over TTL serial
Application Controller	Command Handler	Processes commands from CAM or terminal clients
	Socket Launcher	Launches TCP socket communication threads
	Socket Communication Thread	Communicates with CAM or terminal clients
IPC Manager	Main Thread	Updates main state machine and provisions shared objects

Table 3.2: Processes and Threads in the Interface Application

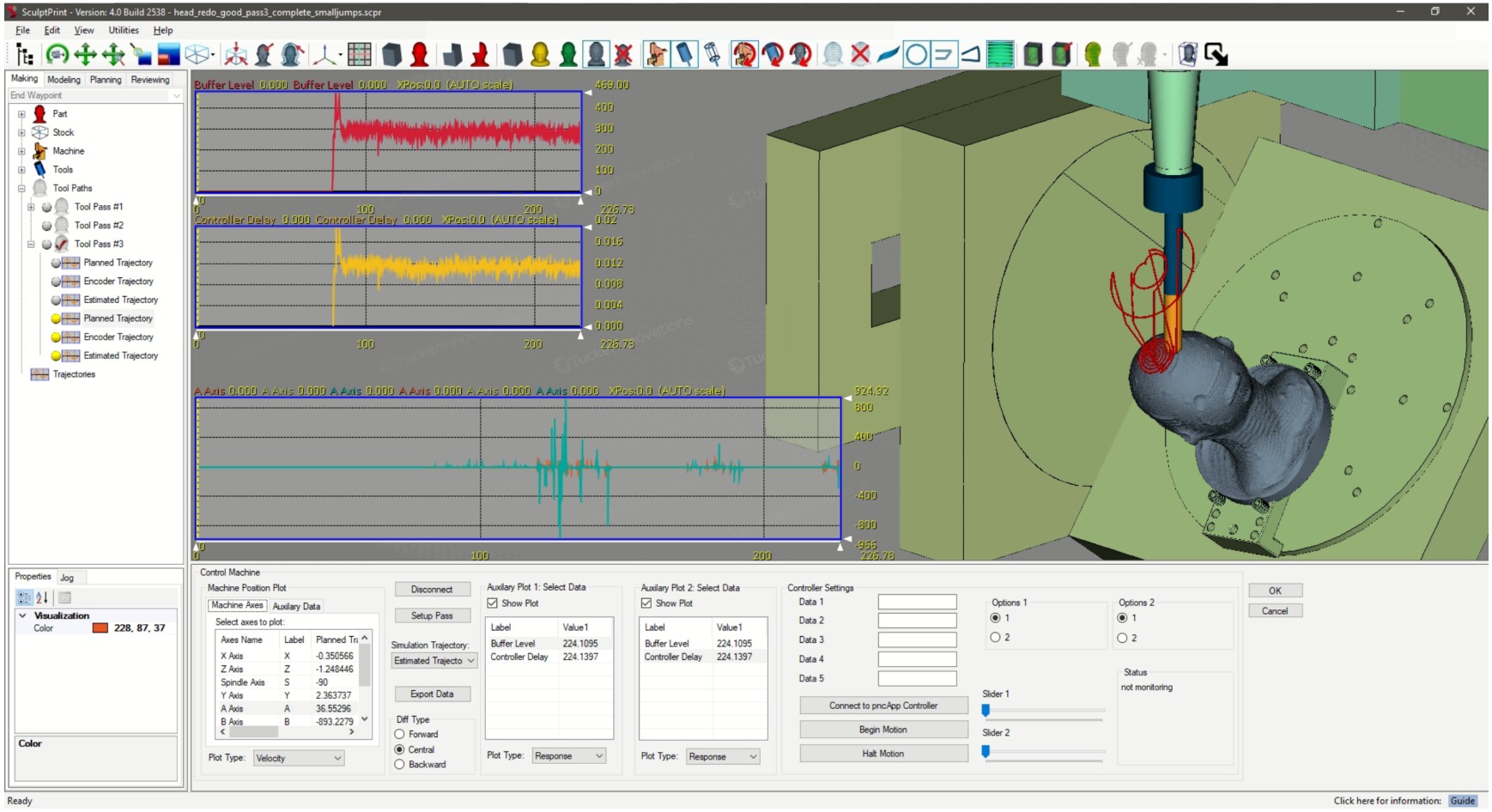


Figure 3.13: User Interface for Machine Control Feature in SculptPrint

CHAPTER 4

CONTROL SYSTEM PERFORMANCE ANALYSIS

Experimental evaluation of the complete machine control system was accomplished using four separate performance metrics. These metrics seek to evaluate the ability of the system to drive the cutting tool over toolpaths generated from voxel models accurately and reliably. The metrics that were assessed are as follows:

1. **Buffer Level Control**: This metric evaluates the effectiveness of the proportional buffer level controller implemented in the interface application used to ensure that servo points are delivered at the correct frequency to the realtime machine control system

2. **Application Interface Latency**: This metric assess the execution time of various key elements of the application interface to determine if they are sufficiently fast to enable reliable control of the machine tool.

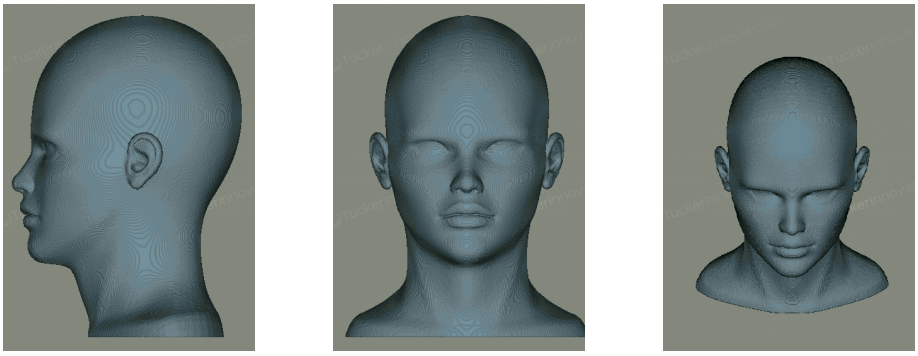
3. **Path Following Capability**: This metric measures the accuracy of the machine tool in following a toolpath in the workpiece reference frame (tool space) using experimental paths whose commanded servo points are compared with stepgenerator feedback and encoder feedback information

4. **Comparison with Traditional Control**: This metric compares the speed of execution of various toolpaths between the direct servo control system and standard G-Code programming

Experimental data used to evaluate these metrics are presented in this chapter, in addition to analysis and discussion of the results.

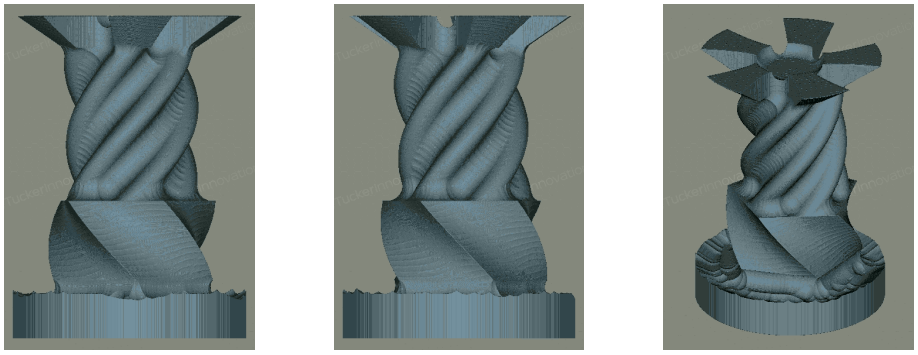
4.1 Test Cases

Toolpaths for two different example parts were developed for this dissertation. These two parts, shown in Figures 4.1 and 4.2, are complex parts that require intricate 5-axis machining and thus served as ideal test cases for evaluation of the machine control architecture. The first part, shown in Figure 4.1, is a human head model that has been utilized in other publications that evaluate SculptPrint's performance as a CAM system. The second model, shown in Figure 4.2, is a twisted candle holder with numerous low accessibility regions that has also been employed in previous publications that evaluated SculptPrint. These two models enable creation of complex toolpaths that necessitate simultaneous 5-axis motion.



(a) Side View (b) Front View (c) Tilted View

Figure 4.1: Human Head Model



(a) Side View (b) Front View (c) Tilted View

Figure 4.2: Candleholder Model

4.1.1 Experimental Toolpaths

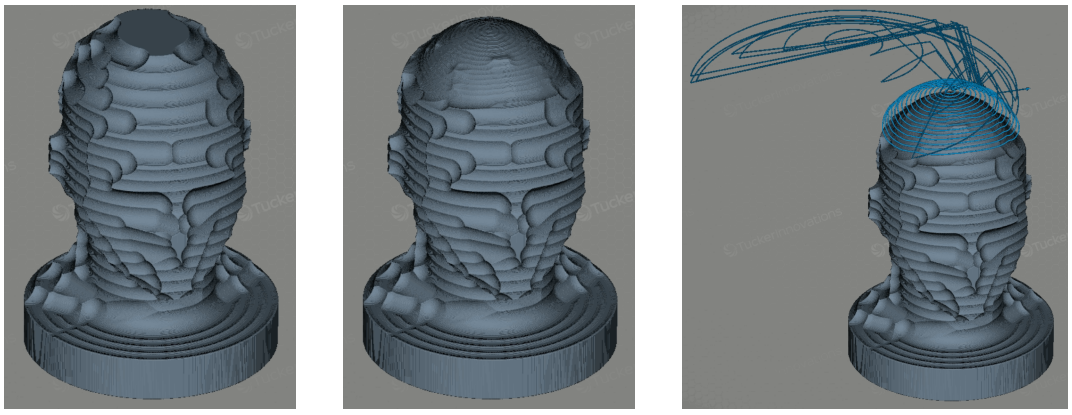
A group of toolpaths was generated for each of these models to serve as experimental input data. Because the roughing phase of the cutting operation (where the workpiece is taken from raw stock to something resembling the final desired geometry in preparation for intricate surface machining) does not require as complex of motion profiles as finishing, the experimental toolpaths used are finishing toolpaths for fine surface machining. Two toolpaths were generated for each part to generate experimental data in both highly accessible smooth regions and tight, low-accessibility areas of the parts. The four toolpaths generated for this thesis are explained below. Each toolpath was created using planar cross sections with a 0.25" diameter ball-nose endmill.

Head Top Toolpath

The first experimental toolpath is a contour toolpath on the top of the human head model. This toolpath was designed to evaluate performance in the highly accessible top region of the part. Figures 4.3(a) and 4.3(b) show the starting and ending volume of the part, respectively. The starting volume represents the state of the part immediately after the roughing process, and the end volume represents the state of the part after the toolpath used to finish the top of the part is complete. The toolpath used to machine the part, as overlaid on the end volume, is shown in Figure 4.3(c). In this Figure, the light blue trace represents the toolpath itself, and the dark blue lines represent retractions that are used to engage and disengage the cutting tool from the workpiece when the tool needs to be repositioned.

Head Bottom Toolpath

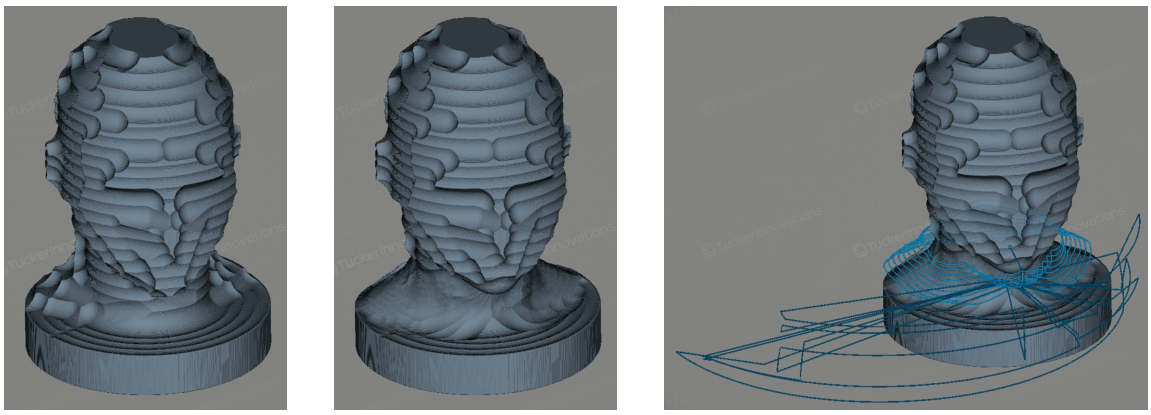
The second experimental toolpath examines the lower accessibility region at the bottom of the head model. This toolpath requires a larger ϕ angle of the cutting tool and contains a number of disjointed movements in the lowest accessibility area under the chin of the head



(a) Starting Volume (b) Ending Volume (c) Toolpath

Figure 4.3: Starting and Ending Volumes for Top Finishing Toolpath for Head Model

model. Figures 4.4(a) and 4.4(b) show the starting and ending volumes for the part model with this toolpath, and Figure 4.4(c) shows the end volume of the part with the toolpath overlaid.



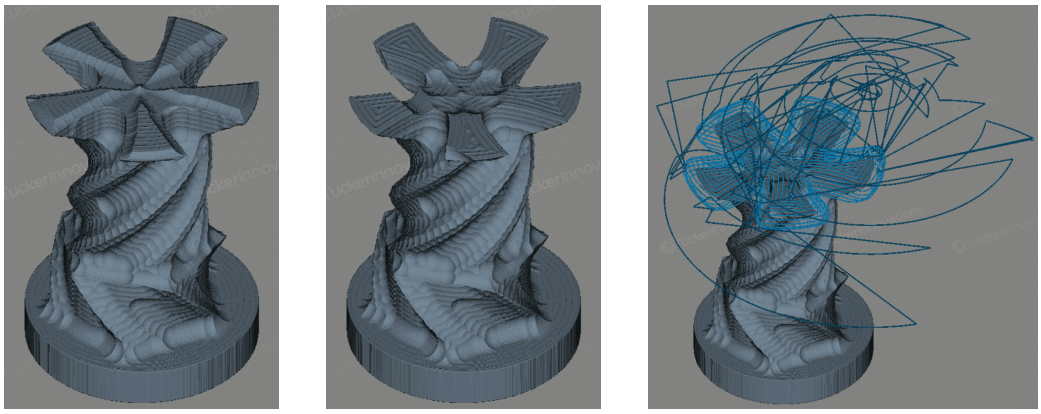
(a) Starting Volume (b) Ending Volume (c) Toolpath

Figure 4.4: Starting and Ending Volumes of Bottom Finishing Toolpath for Head Model

Candleholder Top Toolpath

The third experimental toolpath was designed for the candleholder model, which offers vastly different geometry to the head model. This toolpath was created to finish the top geometry of the model after roughing. Figures 4.5(a) and 4.5(b) show the starting and ending part volumes for the toolpath, and Figure 4.5(c) shows the complete toolpath and

accompanying retractions.

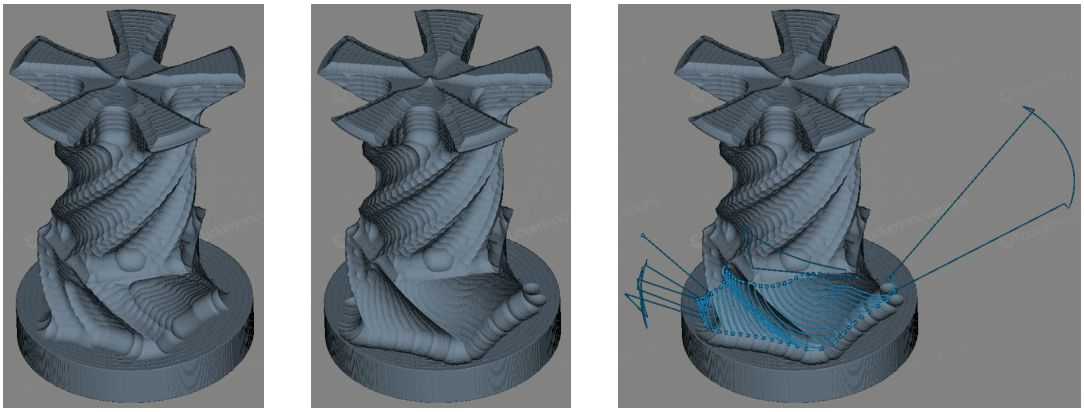


(a) Starting Volume (b) Ending Volume (c) Toolpath

Figure 4.5: Starting and Ending Volumes of Bottom Finishing Toolpath for Head Model

Candleholder Bottom Toolpath

The fourth experimental toolpath was designed to finish a smooth portion of the bottom of the candleholder model. In contrast with the candleholder top toolpath, the bottom toolpath is smoother and operates in a region with higher accessibility, as shown by the smaller number of retractions along the path. The starting and ending volumes for this toolpath are shown in Figures 4.6(a) and 4.6(b), and the complete toolpath with the finished part is shown in Figure 4.6(c).



(a) Starting Volume (b) Ending Volume (c) Toolpath

Figure 4.6: Starting and Ending Volumes of Bottom Finishing Toolpath for Head Model

4.2 Buffer Level Control

This section evaluates the performance of the trajectory buffer level control system that was implemented as interlocking software components that reside within the interface application and MKRSh.

4.2.1 Relevance of Metric

For proper execution of motion profiles, the trajectory buffer must be maintained at an acceptable level to prevent the motion controller from consuming all setpoint commands. As explained in Section 3.2.1, an exhaustion of the trajectory buffer would result in cessation of machine motion, and an overflow of the buffer would shut down the motion controller. The level in the trajectory buffer is maintained using the proportional control architecture described in Section 3.3.1 and shown in Figure 3.7.

Buffer fill level data was recorded for each experimental toolpath and is presented below. For each experiment, the target buffer fill level was 500 points, which corresponds to 500ms of motion at the 1ms servo rate. The trajectory buffer is empty at the beginning of the path, and is charged with 1000 hold position samples before motion begins to avoid unpredictable motion at the start of the path. A larger target buffer level would allow the realtime motion control system to tolerate larger delays in position setpoint transmission, but would also cause the machine to react more slowly if the main control computer issued commands to halt motion. Current buffer fill level readings were transmitted by MKRSh over the TCP socket at an approximate rate of 20 Hz; the readings were interpreted by the feedback handler process and used as a feedback signal to the buffer level controller that resides in the motion control thread of the machine controller process. Statistics for the buffer level measurements from the four experimental toolpaths are shown in Table 4.1.

Toolpath	Mean Buffer Level (ms)	Minimum Buffer Level (ms)	Maximum Buffer Level (ms)
Head Top	234.43	56	934
Head Bottom	244.83	0	1400
Candleholder Top	230.01	0	972
Candleholder Bottom	234.72	94	441

Table 4.1: Statistics for Buffer Level Controller

4.2.2 Head Top Toolpath

The trajectory buffer level for the entirety of the first experimental toolpath is shown in Figure 4.7. As demonstrated by the data, the control system is capable of maintaining the trajectory buffer level in the safe range (between 0 and 2000 points) and achieves a relatively constant average level of 234.43 points (where each point is a single millisecond of motion) in the trajectory buffer. As shown in the buffer level statistics in Table 4.1, the maximum buffer level reading during motion was 934 and the minimum buffer level reading during motion was 56.

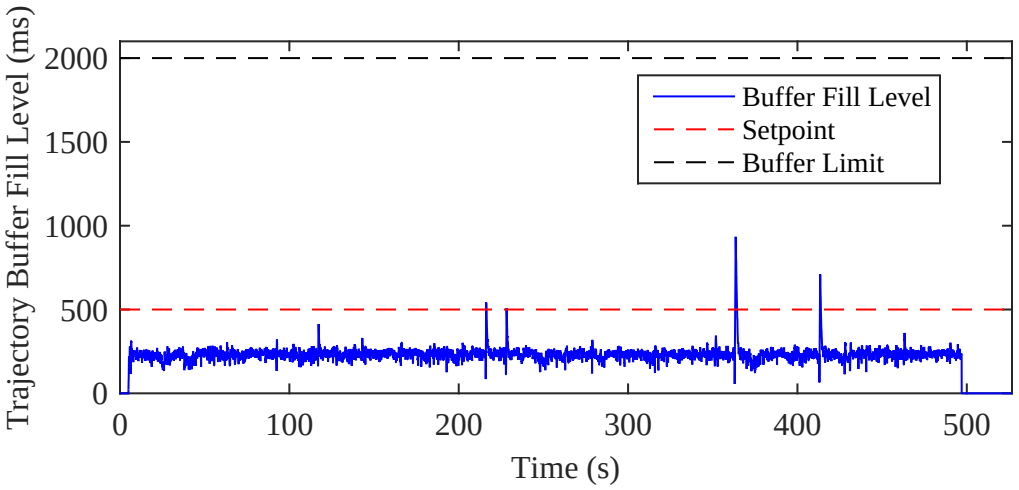


Figure 4.7: Trajectory Buffer Level Readings Obtained During Execution of Head Top Toolpath

4.2.3 Head Bottom Toolpath

Trajectory buffer fill level results for the second experimental toolpath are shown in Figure 4.8. For this experimental toolpath, the buffer level ranges from 0 to 1400, and achieves an average value of 244.83 as presented in Table 4.1. The buffer level controller was not able to maintain the trajectory buffer fill level above zero for the entirety of the toolpath; as shown in Figure 4.8, the buffer runs empty at a few spots along the path. While this behavior is unfortunate, it is difficult to control since the main control computer does not command servo samples at a deterministic rate.

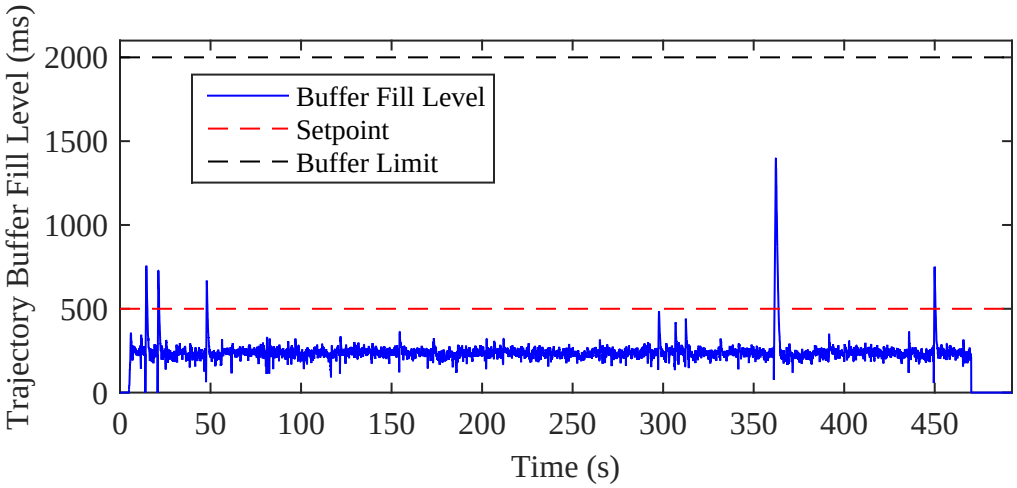


Figure 4.8: Trajectory Buffer Level Readings Obtained During Execution of Head Bottom Toolpath

4.2.4 Candleholder Top Toolpath

The trajectory buffer fill level over the duration of the toolpath used to finish the top of the candleholder is shown in Figure 4.9. From the statistics in Table 4.1, the minimum buffer fill level during path execution was 0, the maximum was 972, and the mean was 230.01. The buffer level also dropped to zero at two points along this toolpath.

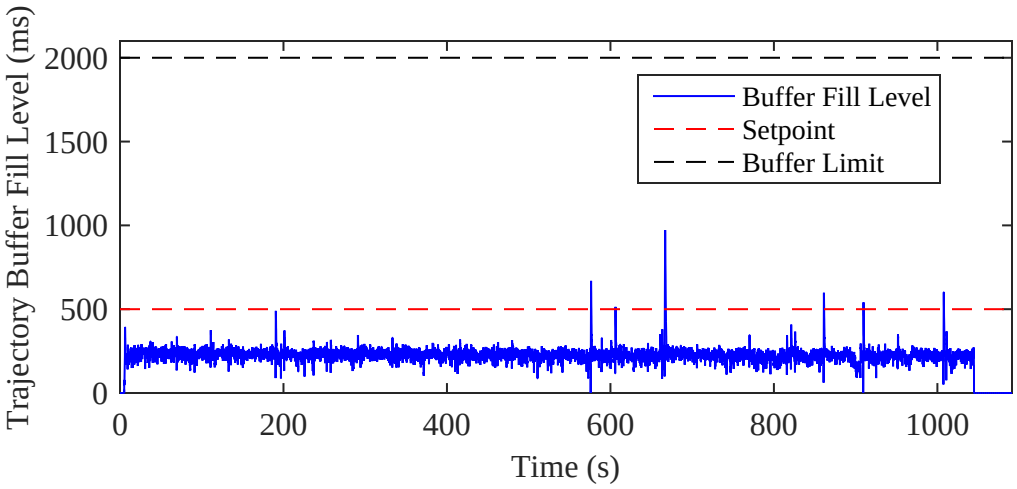


Figure 4.9: Trajectory Buffer Level Readings Obtained During Execution of Candleholder Top Toolpath

4.2.5 Candleholder Bottom Toolpath

Trajectory buffer fill level results for the toolpath used to finish the bottom of the candleholder are shown in Figure 4.10. For this experimental toolpath, the minimum buffer level recorded during path execution was 94, and maximum was 441, and the mean was 234.72.

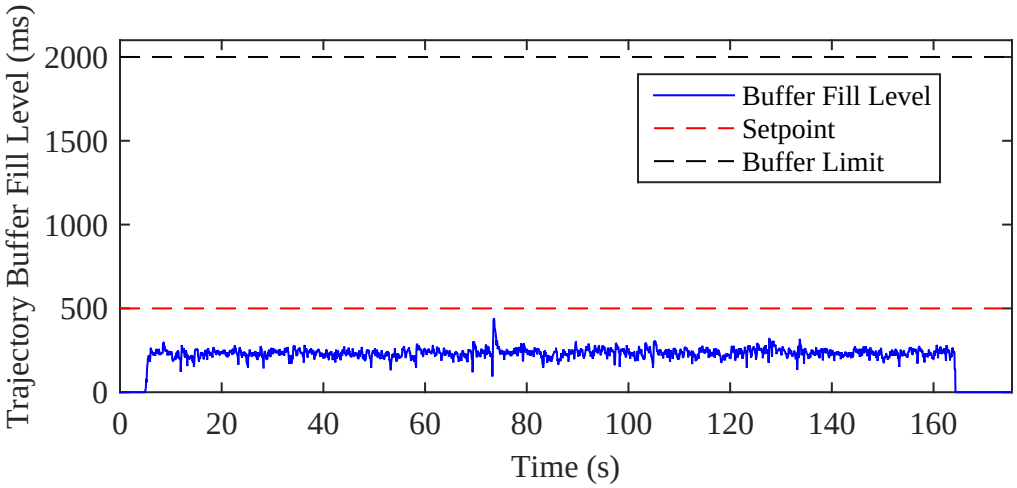


Figure 4.10: Trajectory Buffer Level Readings Obtained During Execution of Candleholder Bottom Toolpath

4.3 Interface Application Performance

This section evaluates the speed of execution of various key components of the interface application that are essential to proper machine operation.

4.3.1 Relevance of Metrics

The interface application running on the main control computer must be able to handle data flow quickly enough to both provide motion commands quickly enough from the CAM system to the machine tool, and to collect process data from the machine and pass it to the CAM system. If the interface application cannot perform all required tasks quickly enough, it will not be able to provide trajectory commands to the machine tool or create accurate realtime visualizations of the machining process. The following results demonstrate that the encoder read frequency is relatively stable throughout each experimental toolpath.

4.3.2 Encoder Read Frequency

Head Top Toolpath

Figure 4.11 shows the frequency with which encoder readings were acquired during execution of the toolpath used to finish the top of the head model. Each reading, which consists of a clock value and the integrated number of encoder counts for each axis, is acquired by the device interface process (described in Section 3.3.3) from the USB connection to the encoder interface board that was described in Section 2.6. The read frequency is calculated by taking the inverse of the difference in clock values between each reading. The blue trace is the instantaneous acquisition frequency measured by the interface application, and the red trace is a 200-sample moving average of the acquisition frequency.

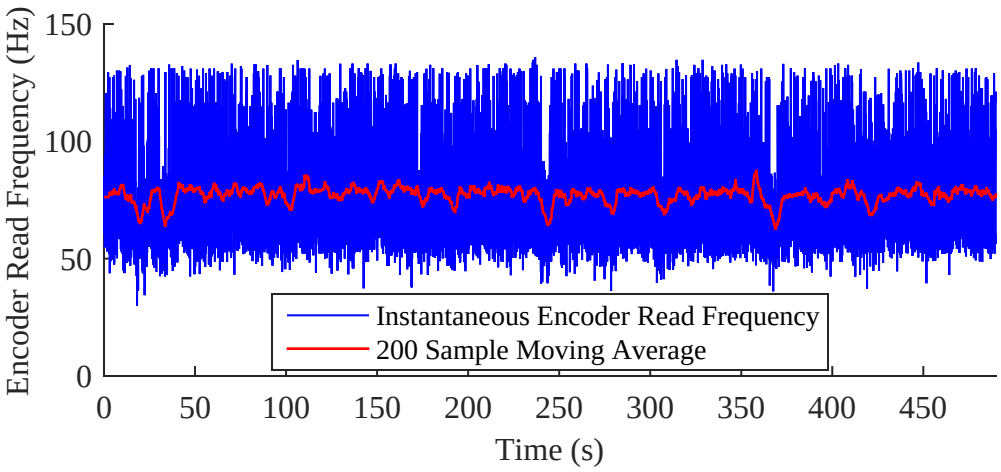


Figure 4.11: Encoder Read Frequency During Execution of Head Top Toolpath

Head Bottom Toolpath

Similar results were obtained for the toolpath used to finish the bottom of the head model. The instantaneous acquisition frequency and the 200-sample moving average of the acquisition frequency are shown in Figure 4.12

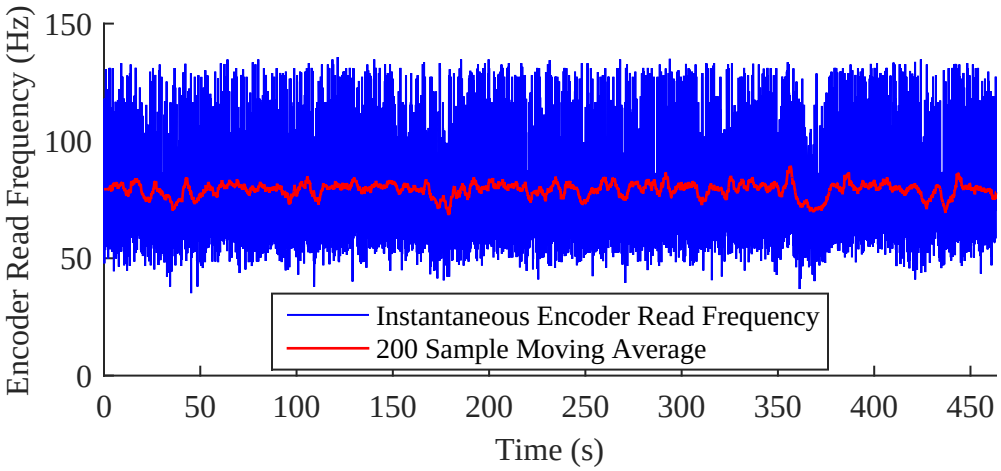


Figure 4.12: Encoder Read Frequency During Execution of Head Bottom Toolpath

Candleholder Top Toolpath

Similar to the results obtained above, the encoder read frequency data for the toolpath used to finish the top of the candleholder is shown in Figure 4.13.

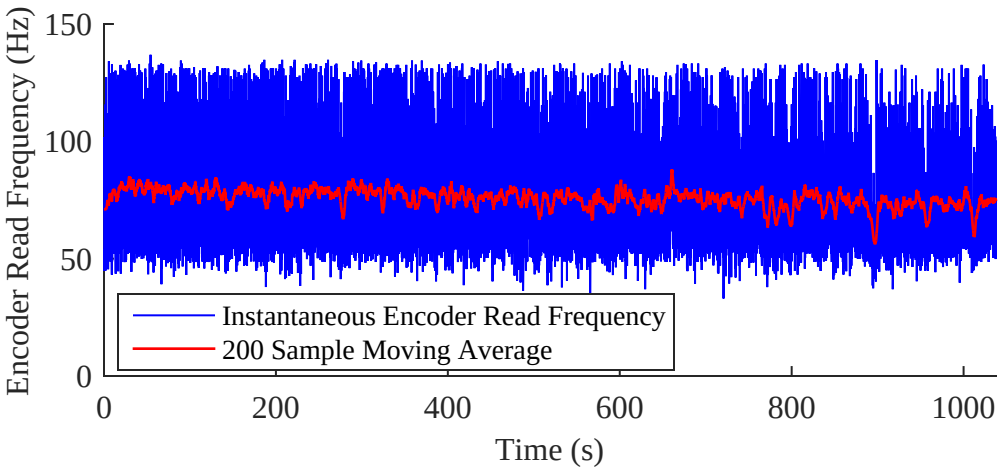


Figure 4.13: Encoder Read Frequency During Execution of Candleholder Top Toolpath

Candleholder Bottom Toolpath

Similar to the results obtained above, the encoder read frequency data for the toolpath used to finish the bottom of the candleholder is shown in Figure 4.14.

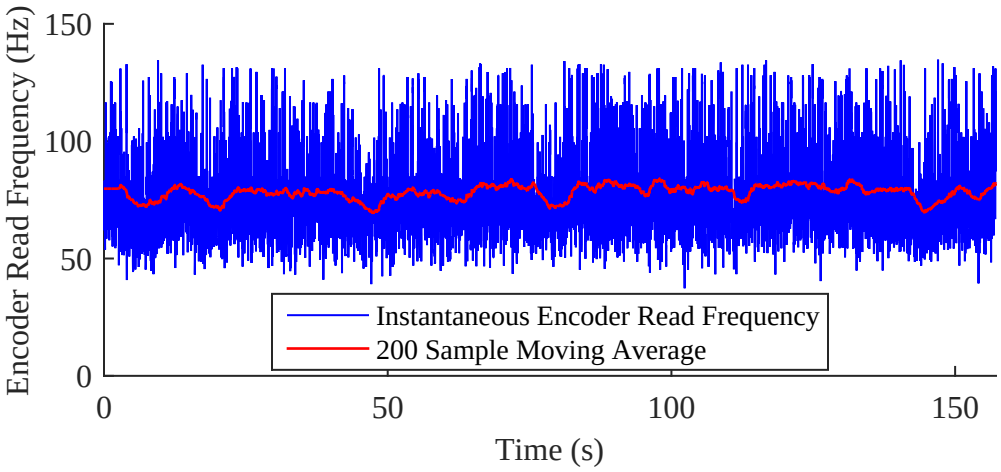


Figure 4.14: Encoder Read Frequency During Execution of Candleholder Bottom Toolpath

4.3.3 Servo Point Transmission Delay

The interface application must be capable of supplying servo loop commands to the machine tool at a high enough frequency that the motion controller on the BBB does not run out of points to execute. The servo commands are sent over the TCP socket in packets as

described in Section 3.2.2 using a controllable delay between transmissions, as explained in Section 3.3.1. The delay is controlled with the proportional controller described in Equation 3.2 whose error signal is calculated using buffer level readings transmitted by MKRSh to the feedback handler process.

Head Top Toolpath

The transmission delays used in the buffer level controller during execution of the toolpath used to finish the top of the head model are shown in Figure 4.15. The raw data is shown as a blue line. A zero phase finite impulse response filter was applied to the data to enable easier visualization of trends, and the filtered data is shown in red. The experimental data demonstrates that the interface application is generally able to keep up with the servo point transmission frequency required by the motion controller, as the transmission delay usually remains above zero. A commanded delay of zero indicates that the saturation operator in Equation 3.2 is active because the current buffer level reading is too low. If the commanded delay remains at zero for too long, the interface application is incapable of providing servo setpoints to the BBB fast enough and the trajectory buffer will empty. Delays of zero at the beginning of the path correspond to initial charging of the trajectory buffer. The noise in commanded delays is caused by variability in execution time on the main control computer.

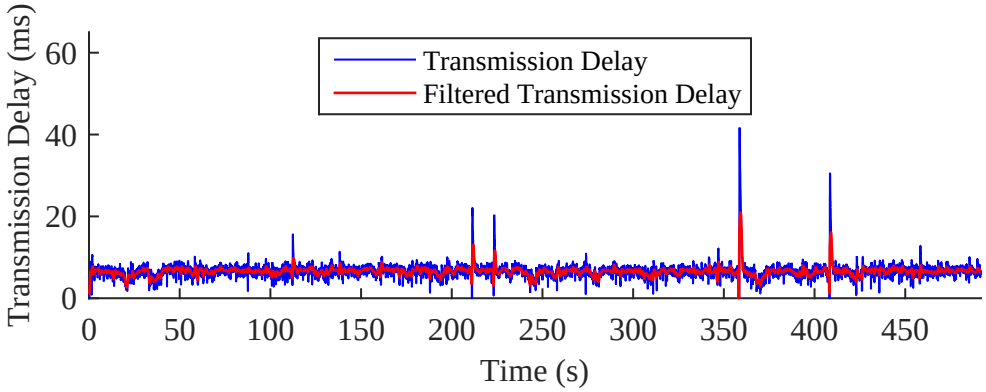


Figure 4.15: Servo Point Transmission Delay During Execution of Head Top Toolpath

Head Bottom Toolpath

Experimental values of transmission delays during execution of the toolpath used to finish the bottom of the head model are shown in Figure 4.16. Again, raw data are shown in blue and filtered data are shown in red.

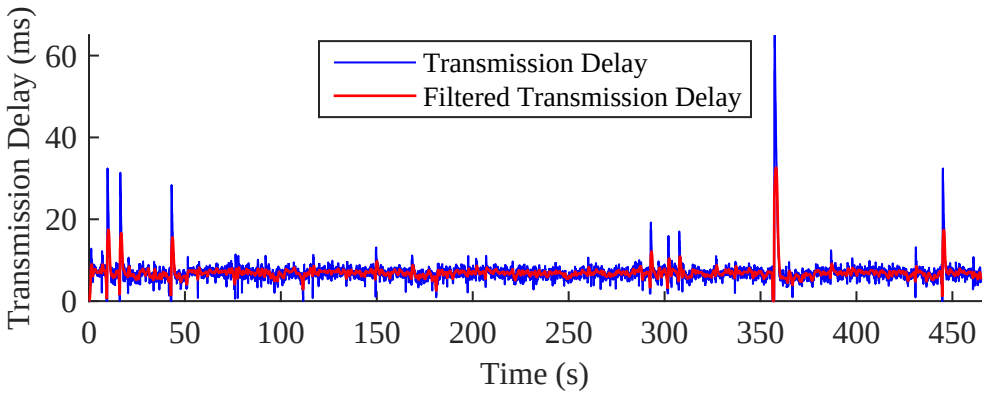


Figure 4.16: Servo Point Transmission Delay During Execution of Head Bottom Toolpath

Candleholder Top Toolpath

Similar to the results above, transmission delay data gathered during execution of the toolpath used to finish the top of the candleholder are shown in Figure 4.17.

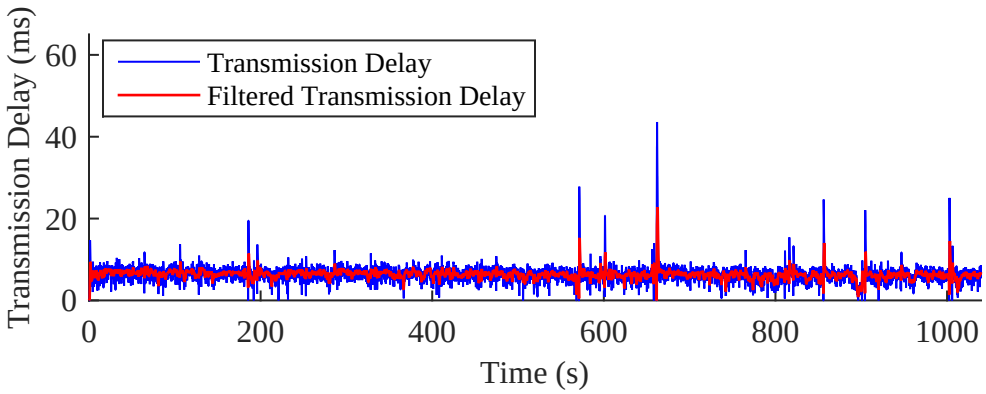


Figure 4.17: Servo Point Transmission Delay During Execution of Candleholder Top Toolpath

Candleholder Bottom Toolpath

Similar to the results above, transmission delay data gathered during execution of the toolpath used to finish the bottom of the candleholder are shown in Figure 4.18.

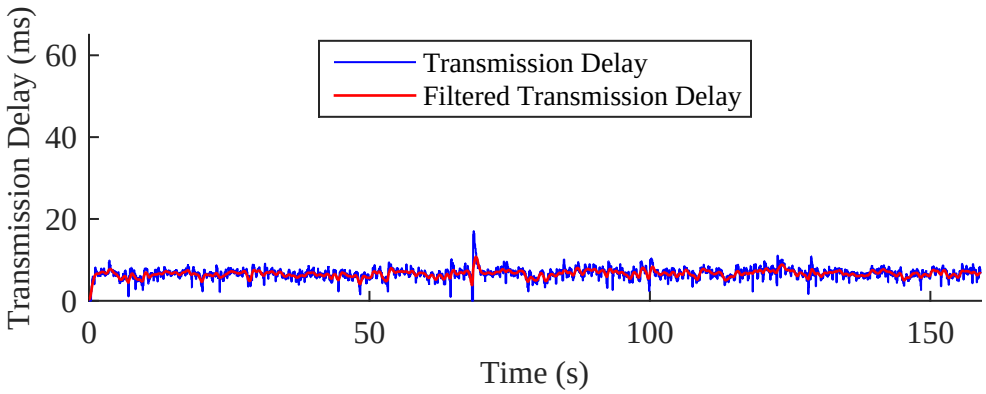


Figure 4.18: Servo Point Transmission Delay During Execution of Candleholder Bottom Toolpath

4.3.4 Database Push/Pull Performance

Performance of the database process is also essential to proper machine operation, as it is responsible for both storing all process data collected during machining and for updating the main state machine maintained in the interface application. During operation, the database is continuously responding to push and pull commands (corresponding to storage or retrieval requests for specific sets of data) by other processes in the application. Processes issuing retrieval requests to the database must wait for the database to return the requested data before continuing their execution, so timely response of the database is essential to proper functioning of the interface application. Timely operation of storage requests is less critical, as any storage request is simply issued to the database and the requestor continues execution without waiting for the request to complete. The average number of retrieval and storate requests that the database had to process each second for each toolpath are shown in Table 4.2. These data are useful to put the database request execution times into perspective: if the database cannot process requests fast enough to meet

the number of requests per second shown in the Table, the database will fall behind and not function properly.

Toolpath	Retrieval Request Frequency (Hz)	Storage Request Frequency (Hz)
Head Top	66.461	37.881
Head Bottom	68.622	39.136
Candleholder Top	65.032	37.270
Candleholder Bottom	74.500	41.067

Table 4.2: Database Request Frequency for Experimental Toolpaths

Head Top Toolpath

The pull and push request execution times recorded during execution of the toolpath used to finish the top of the head model are shown in Figure 4.19. The top plot shows the pull duration as a function of the request count in blue, and a 20-sample moving average of the pull duration in red. The bottom plot shows the instantaneous push duration as a function of the request count in blue, and a 20-sample moving average of the push execution duration in red. The moving average reveals that the pull duration does not demonstrate an upward trend throughout execution of the path, but the push duration does. However, the moving average demonstrates that the database processes requests fast enough to keep up with the 66.461 Hz retrieval request and 37.881 Hz storage request frequencies from Table 4.2.

Head Bottom Toolpath

Similar results were obtained for database performance during execution of the toolpath used to finish the bottom of the head model. Figure 4.20 shows pull duration in the top plot and push duration in the bottom plot with moving averages overlaid. Again, the moving average demonstrates that the database processes requests fast enough to keep up with the 68.622 Hz retrieval request and 39.136 Hz storage request frequencies from Table 4.2.

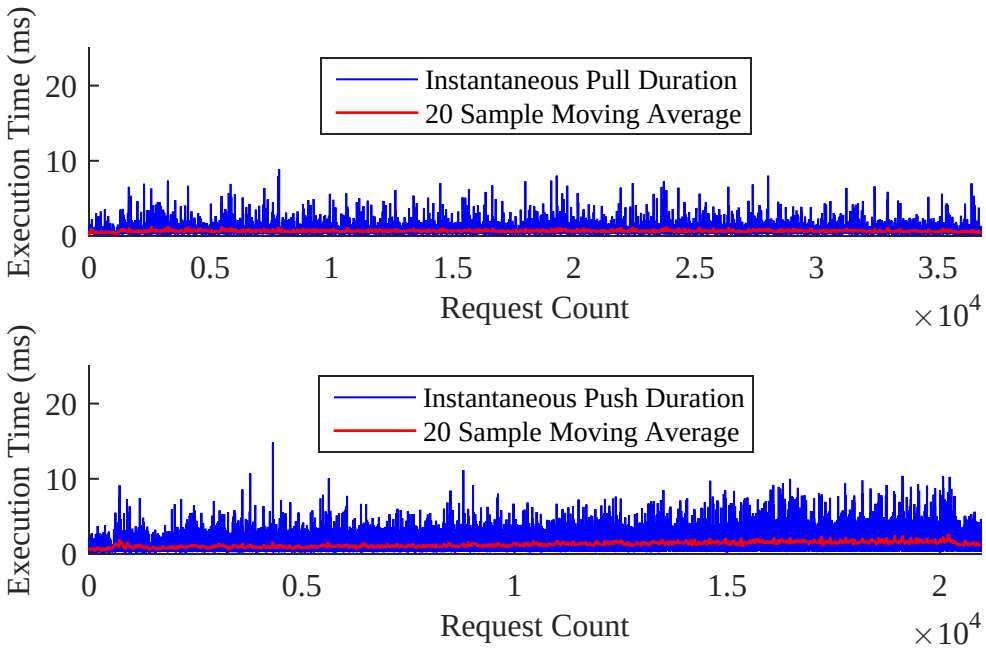


Figure 4.19: Database Performance During Execution of Head Top Toolpath

Candleholder Top Toolpath

Since the toolpath used to finish the top of the candleholder is the longest of the experimental toolpaths, the upward trend in database push command execution times is expected to be more pronounced. Figure 4.21 shows the experimental results, where the raw database command execution times are shown in blue, and the moving averages of the pull and push times are shown in red. Although the upward trend in storage request processing time is more pronounced for this toolpath, the moving average demonstrates that the database processes requests fast enough to keep up with the 65.032 Hz retrieval request and 37.270 Hz storage request frequencies from Table 4.2.

Candleholder Bottom Toolpath

The toolpath used to finish the bottom of the candleholder model is the shortest of the experimental toolpaths, so database performance is expected to be faster during execution of the path since the database storage area in the main control computer is smaller. Database

Figure 4.20: Database Performance During Execution of Head Bottom Toolpath

request processing performance is shown in Figure 4.22 as is the case in previous sections. The moving average demonstrates that the database processes requests fast enough to keep up with the 74.500 Hz retrieval request and 41.067 Hz storage request frequencies from Table 4.2.

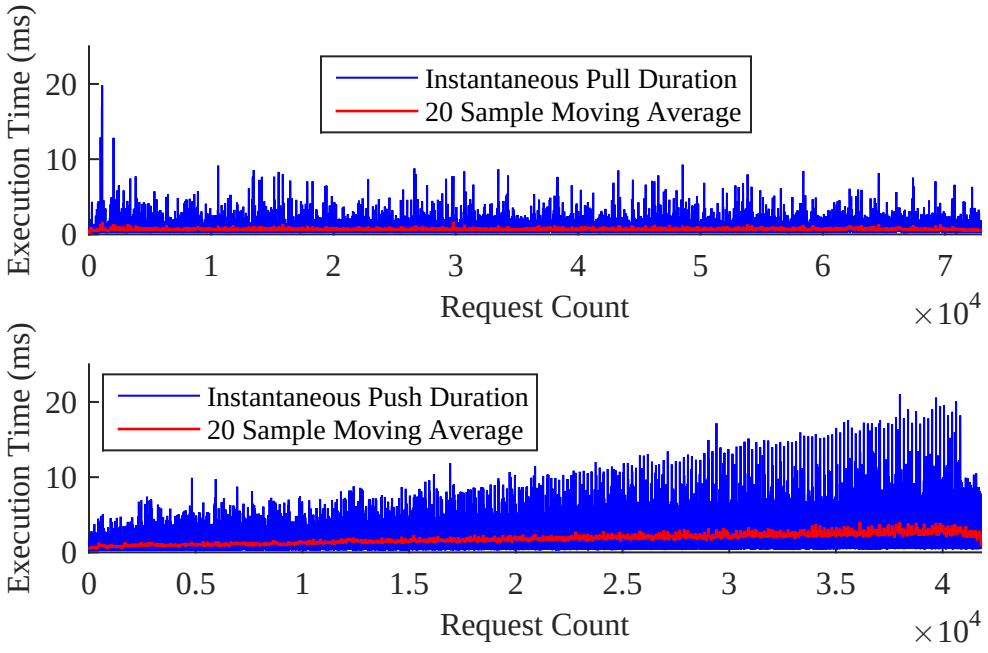


Figure 4.21: Database Performance During Execution of Candleholder Top Toolpath

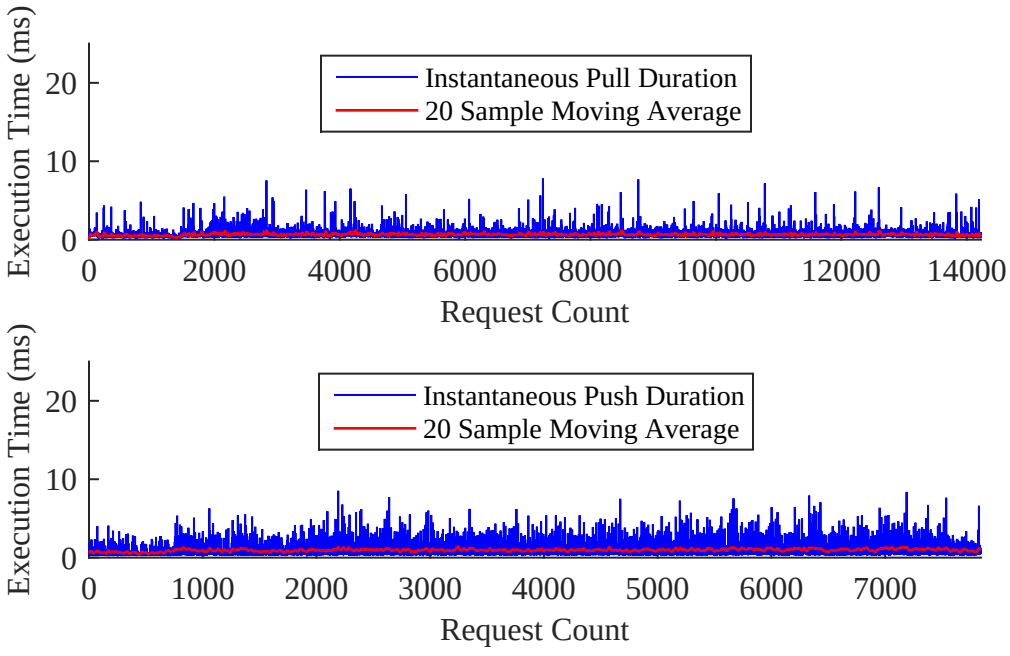


Figure 4.22: Database Performance During Execution of Candleholder Top Toolpath

4.4 Path Following

This section evaluates the capability of the motion control system to follow commanded positions and velocities that are given by the cloud trajectory planning system.

4.4.1 Relevance of Metric

Any effective motion control system should be able to follow a prescribed path with minimal error. The experimental toolpaths described in Section 4.1.1 were executed on the machine tool developed for this dissertation and used as evaluation data of the complete motion control system that consists of the python interface application and the custom RT and non-RT components of Machinekit. The complete motion control system hardware and accompanying software enables the execution of pre-planned trajectories that are sampled at the one millisecond servo rate of the motion control system, and also provides a framework for high-frequency collection of both estimated and actual axis position data. The estimated axis position is determined simply by counting the number of step commands that are generated by the PRU on the BBB for each axis. As a result, the estimated axis position can be interchangeably referred to as the step generator (or stepgen) position. The actual axis position is determined by a rotary encoder that is attached to each axis actuator, and can be interchangeably referred to as the encoder position. The following sections present planned, estimated, and actual axis positions and velocities for each toolpath to demonstrate that the machine tool is capable of following the commanded motion profiles. Additionally, 3D visualizations of the planned and realized toolpaths in the workpiece coordinate system (obtained by transformation of axis positions using Equation 1.6) are shown using the SculptPrint interface that was developed for this work.

Approximately one gigabyte of data was collected from the machine tool during experimental validation for the four toolpaths. As a result, even though the toolpaths are on the order of minutes in length, visualization of all collected data is not possible in this disser-

tation. As a result, notable areas of the paths are presented separately from the complete axis data plots to highlight path characteristics. The total execution time of each toolpath is presented in Table 4.3.

Toolpath	Total Execution Time (s)
Head Top	492.471
Head Bottom	465.630
Candleholder Top	1042.701
Candleholder Bottom	158.788

Table 4.3: Execution Time for Experimental Toolpaths

Table 4.4 presents trajectory error statistics used to evaluate the path following metric. The reported trajectory error E_t is calculated using the norm of the vector between each actual toolpath point (as measured by the axis encoders) and the corresponding commanded point (i.e., the servo sample in the trajectory buffer), as described in Equation 4.1,

$$E_t(t) = \sqrt{[x_C(t) - x_A(t)]^2 + [y_C(t) - y_A(t)]^2 + [z_C(t) - z_A(t)]^2} \tag{4.1}$$

where x_C, y_C, and z_C are the commanded X, Y, and Z positions in tool space, respectively; x_A, y_A, and z_A are the actual X, Y, and Z positions in tool space as measured by the axis encoders, respectively; and t is the query time, which ranges from 0 to the total toolpath execution time from Table 4.3. Thus, this metric only evaluates the *translational* error of the toolpath, and does not directly assess the complete *pose* error. It is worth mentioning that, because the research machine tool has two rotational degrees of freedom (i.e., A and B axes), errors in rotary axis position do affect translational error upon transformation to WPC.

4.4.2 Head Top Toolpath

This section presents visualizations of of raw joint space data that was collected from the machine tool during execution of the toolpath for finishing the top of the head model, in addition to the corresponding tool space data (referred to as workpiece coordinate system,

Toolpath	Range (μm)	Mean (μm)	St. Dev. (μm)
Head Top	0 249.2	50.5	29.2
Head Bottom	0 232.9	35.0	28.5
Candleholder Top	0 367.8	54.7	37.1
Candleholder Bottom	0 211.1	40.1	29.6

Table 4.4: Trajectory Error Statistics for Experimental Toolpaths

or WPC, data) that was obtained from transformation of the joint space data using the FKT.

Joint Space Data

The axis position and velocity progressions for the head top toolpath for each of the five axes of the machine tool are shown in Figures 4.23-4.35. The commanded position, which is shown as a blue dot-dash line, corresponds to the planned trajectory that was obtained from the cloud trajectory planner during operation. The red dashed line corresponds to the estimated axis position, which is obtained from the step generator. The solid black line corresponds to axis position obtained from the encoders. Inspection of the full data in Figures 4.23, 4.26, 4.29, 4.32, and 4.35 reveals that the estimated and actual axis positions closely track the commanded position if viewed at the full time range of the path. The estimated and actual axis velocities also closely track the commanded velocities, though as expected, the velocity data (obtained by numerical differentiation of the position data) contains a fair amount of noise. Twenty second detail views of dynamic portions of the position and velocity data are shown in Figures 4.24, 4.27, 4.30, 4.33, and 4.36. Three second detail views are shown in Figures 4.25, 4.28, 4.31, 4.34, and 4.37. The detail views reveal a slight phase shift between the commanded positions and velocities and the actual positions and velocities, which is caused by the buffering system created to absorb non-deterministic network latency between the main control computer and the realtime motion controller. The lag at each point time is equivalent to the number of points in the trajectory

buffer of the motion controller. Experimental data demonstrate that both the estimated and actual axis positions and velocities faithfully track the commanded values.

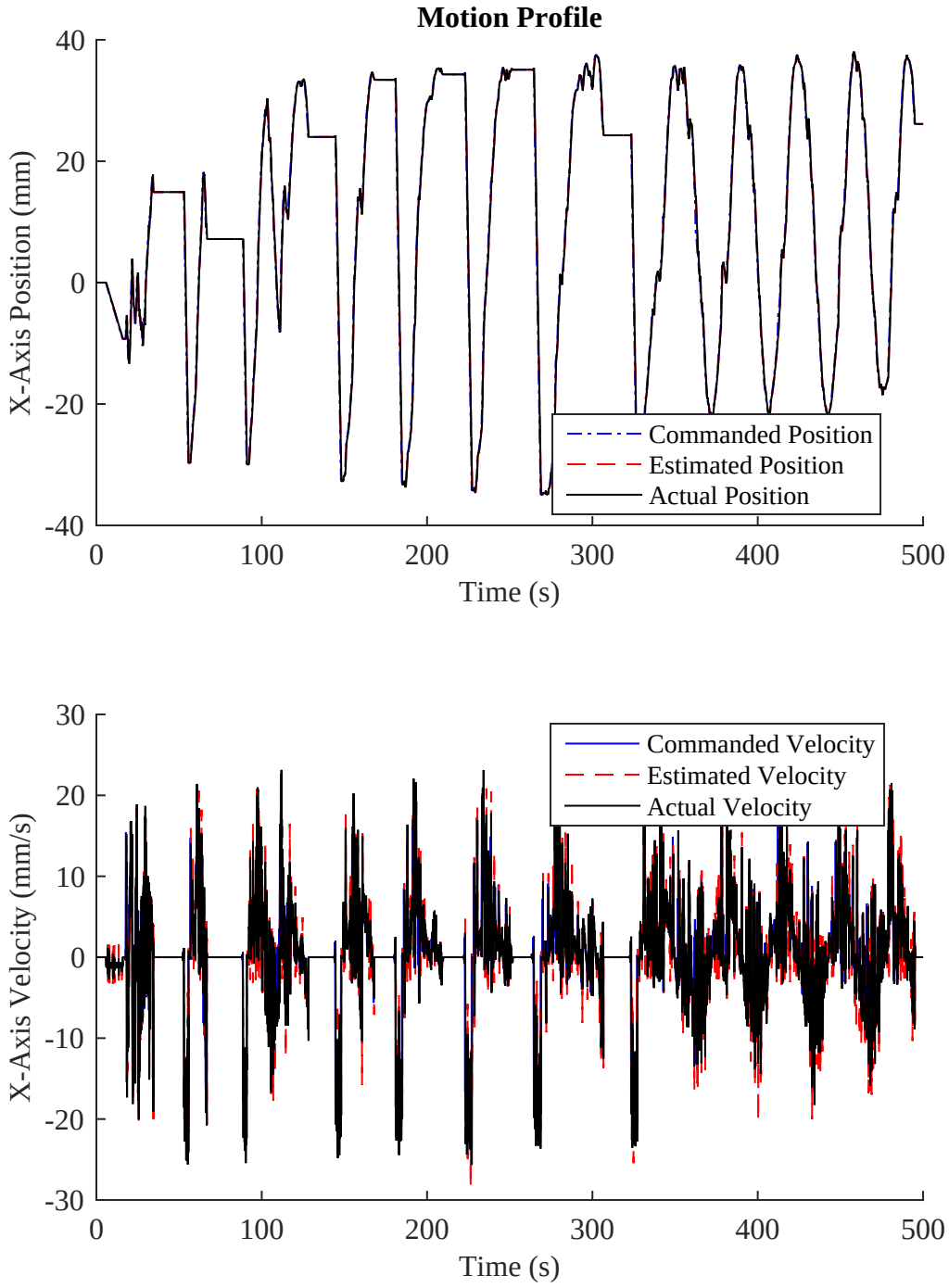


Figure 4.23: X-axis Position and Velocity Progression for Head Top Toolpath

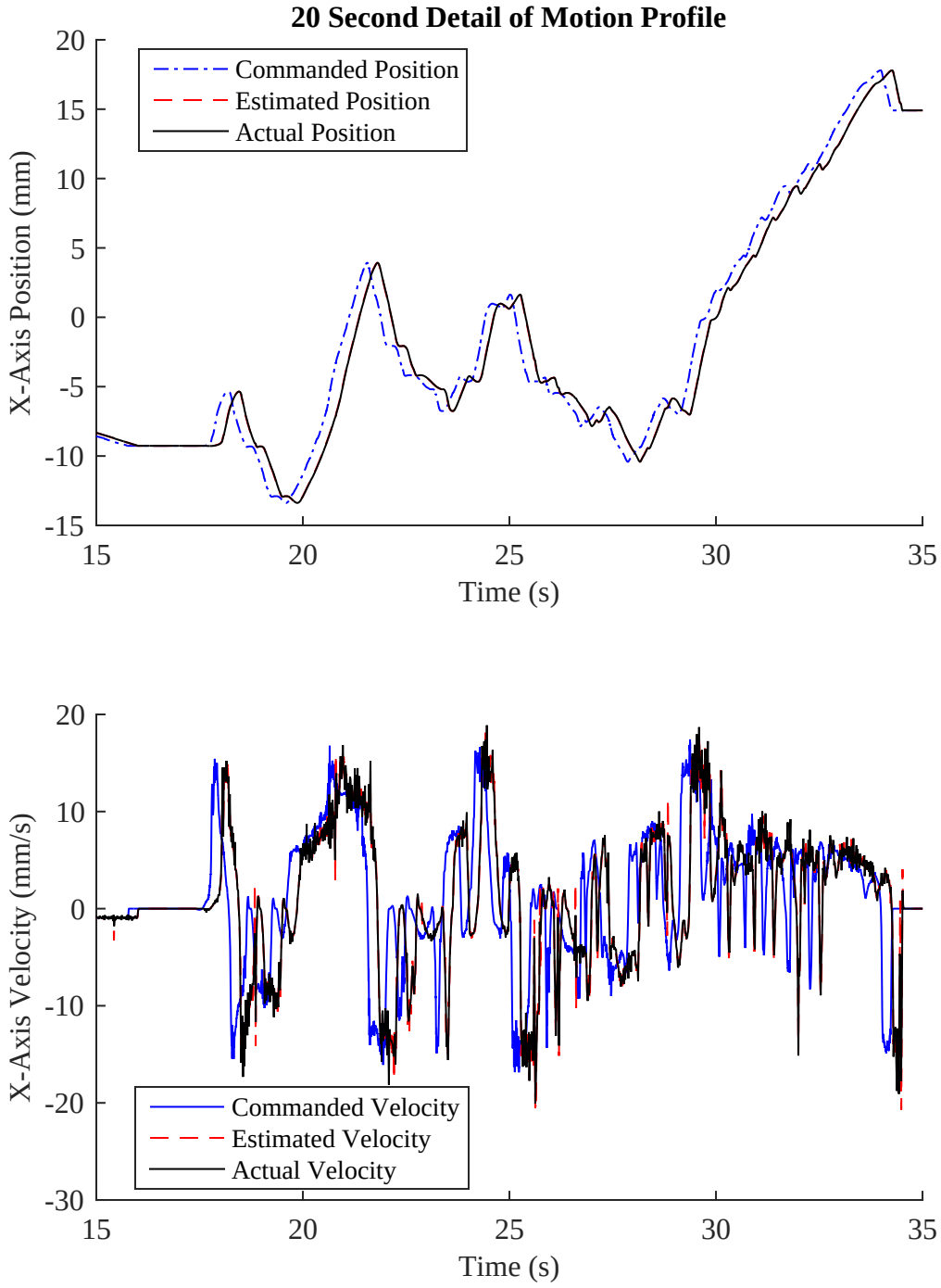


Figure 4.24: 20 Second X-axis Position and Velocity Progression for Head Top Toolpath

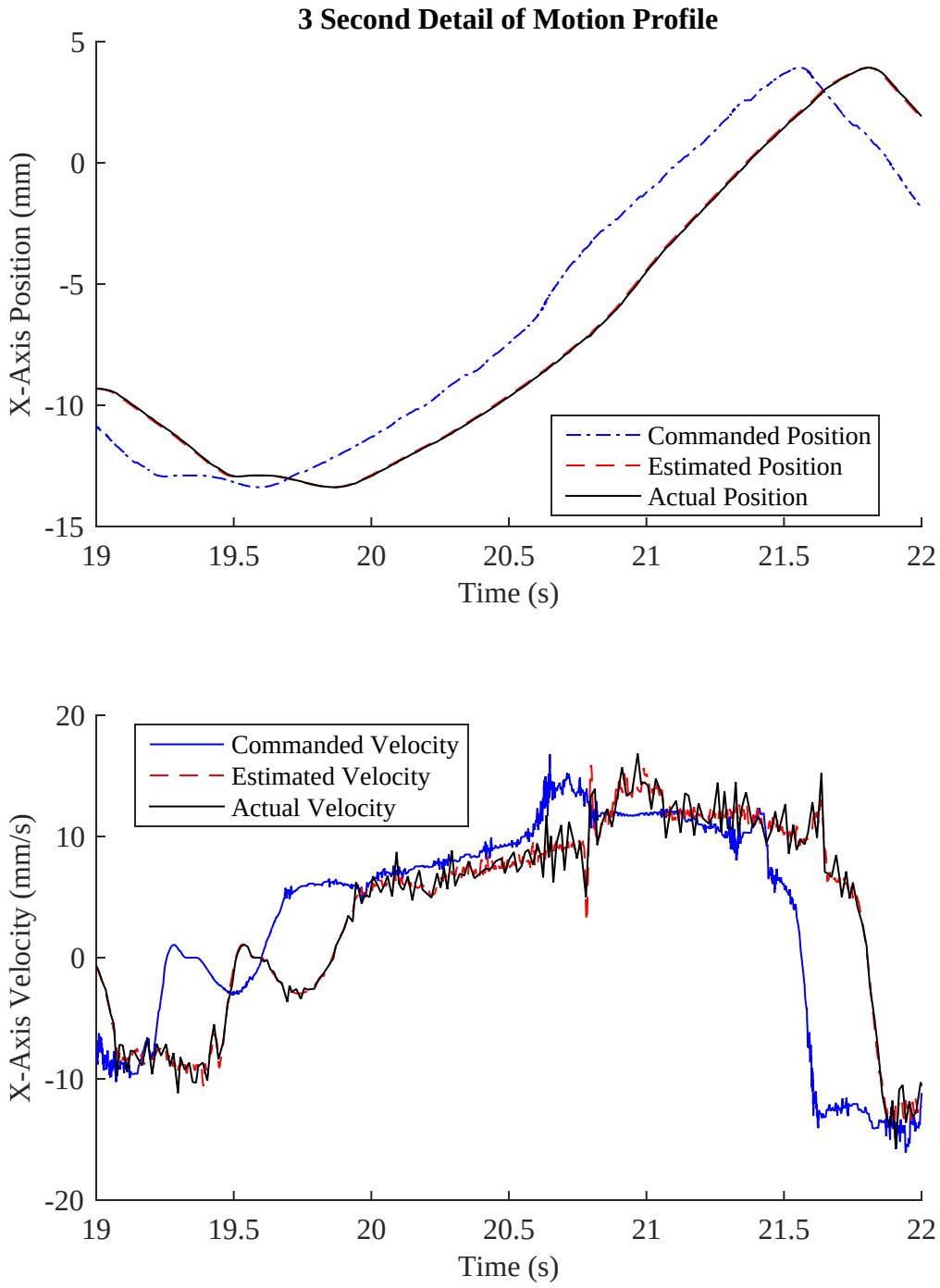


Figure 4.25: 3 Second X-axis Position and Velocity Progression for Head Top Toolpath

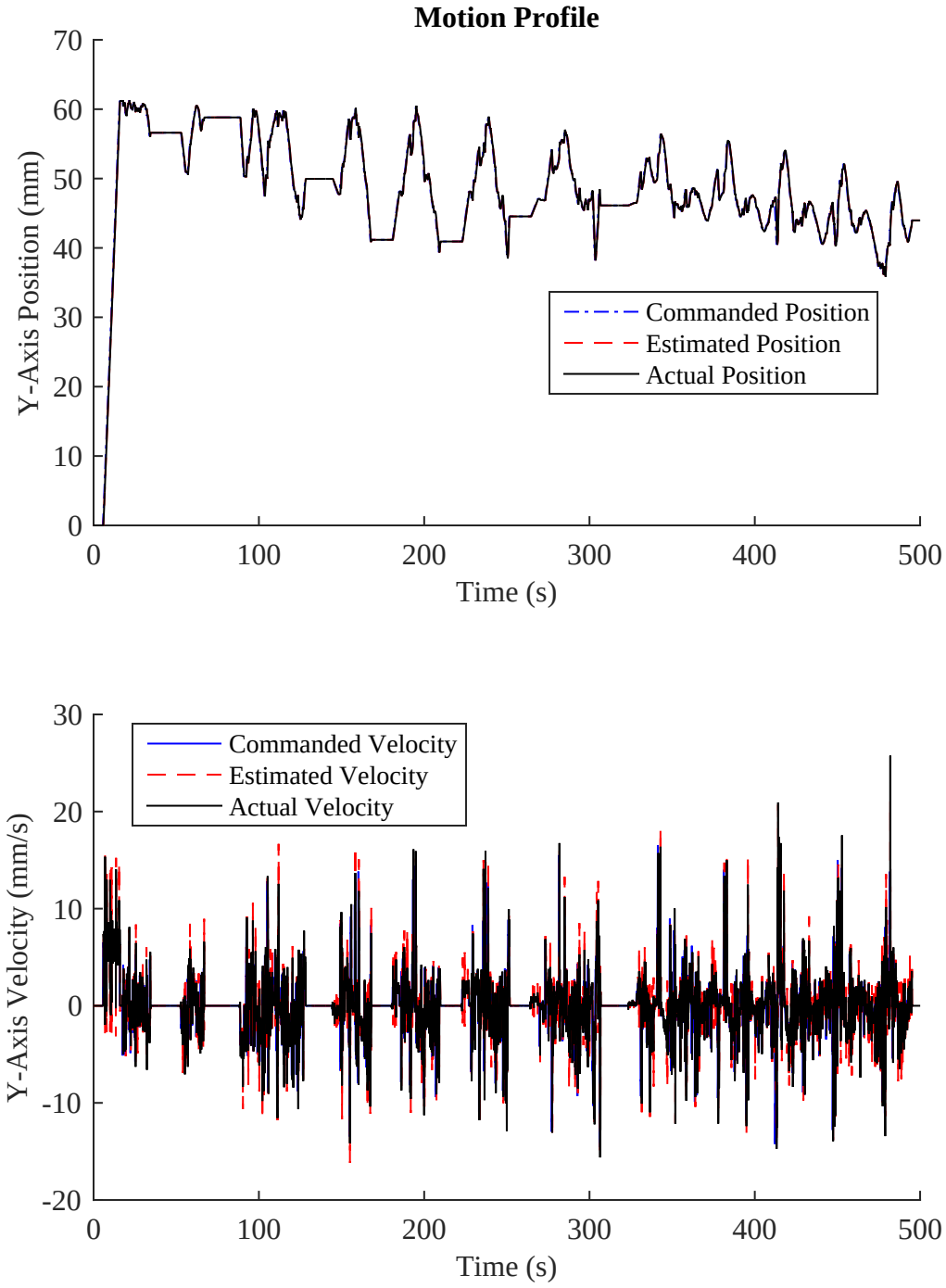


Figure 4.26: Y-axis Position and Velocity Progression for Head Top Toolpath

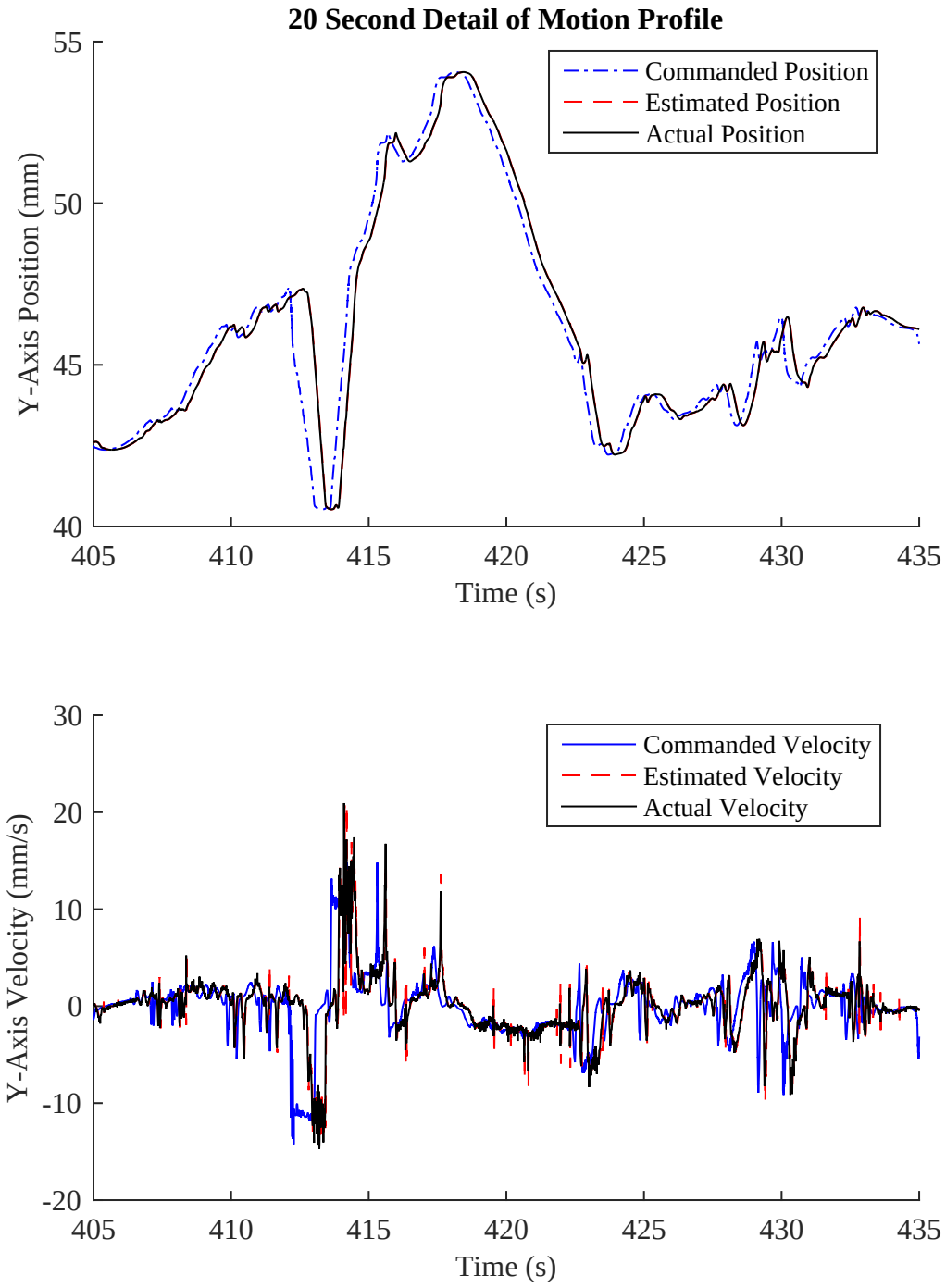


Figure 4.27: 20 Second Y-axis Position and Velocity Progression for Head Top Toolpath

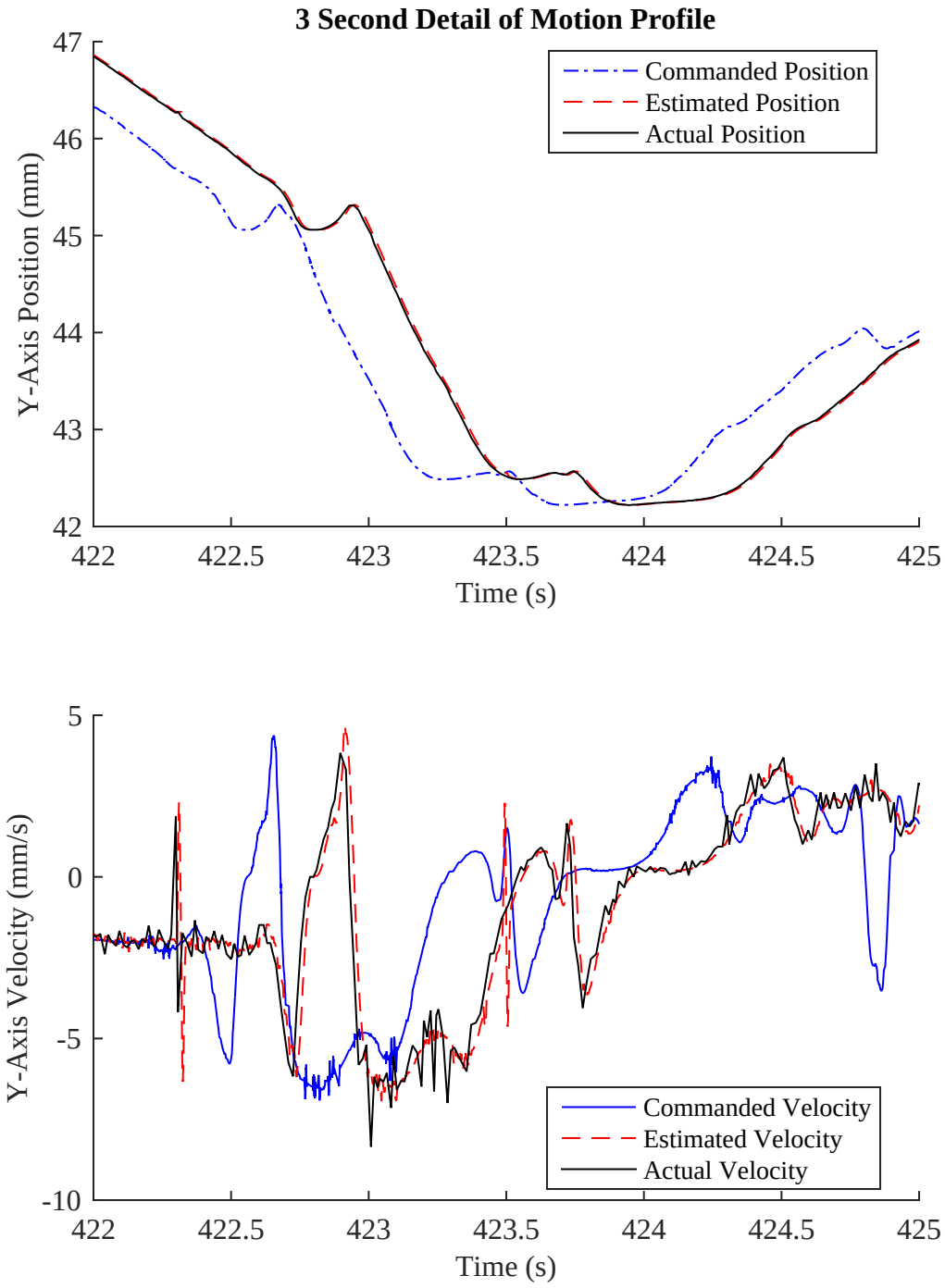


Figure 4.28: 3 Second Y-axis Position and Velocity Progression for Head Top Toolpath

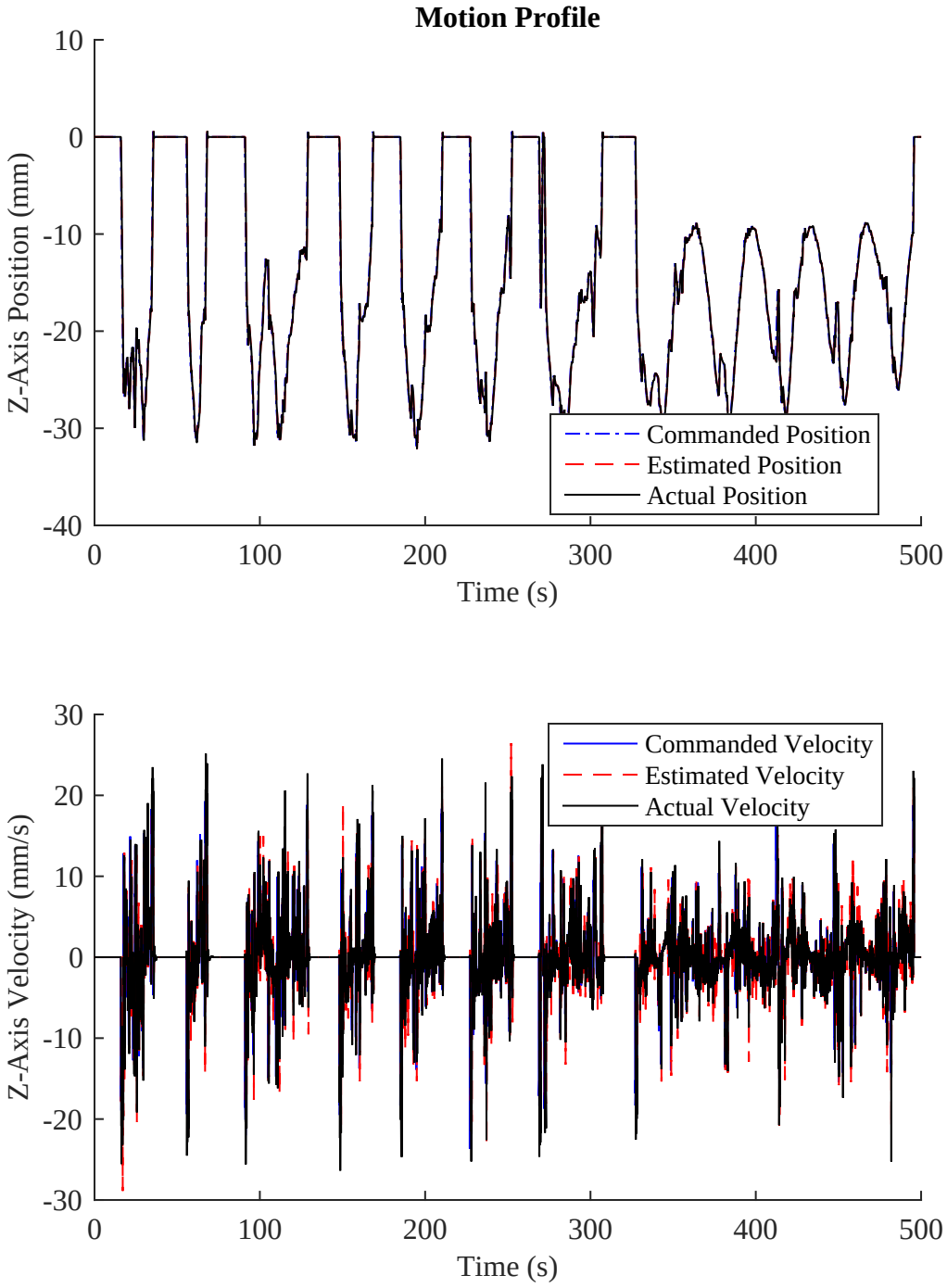


Figure 4.29: Z-axis Position and Velocity Progression for Head Top Toolpath

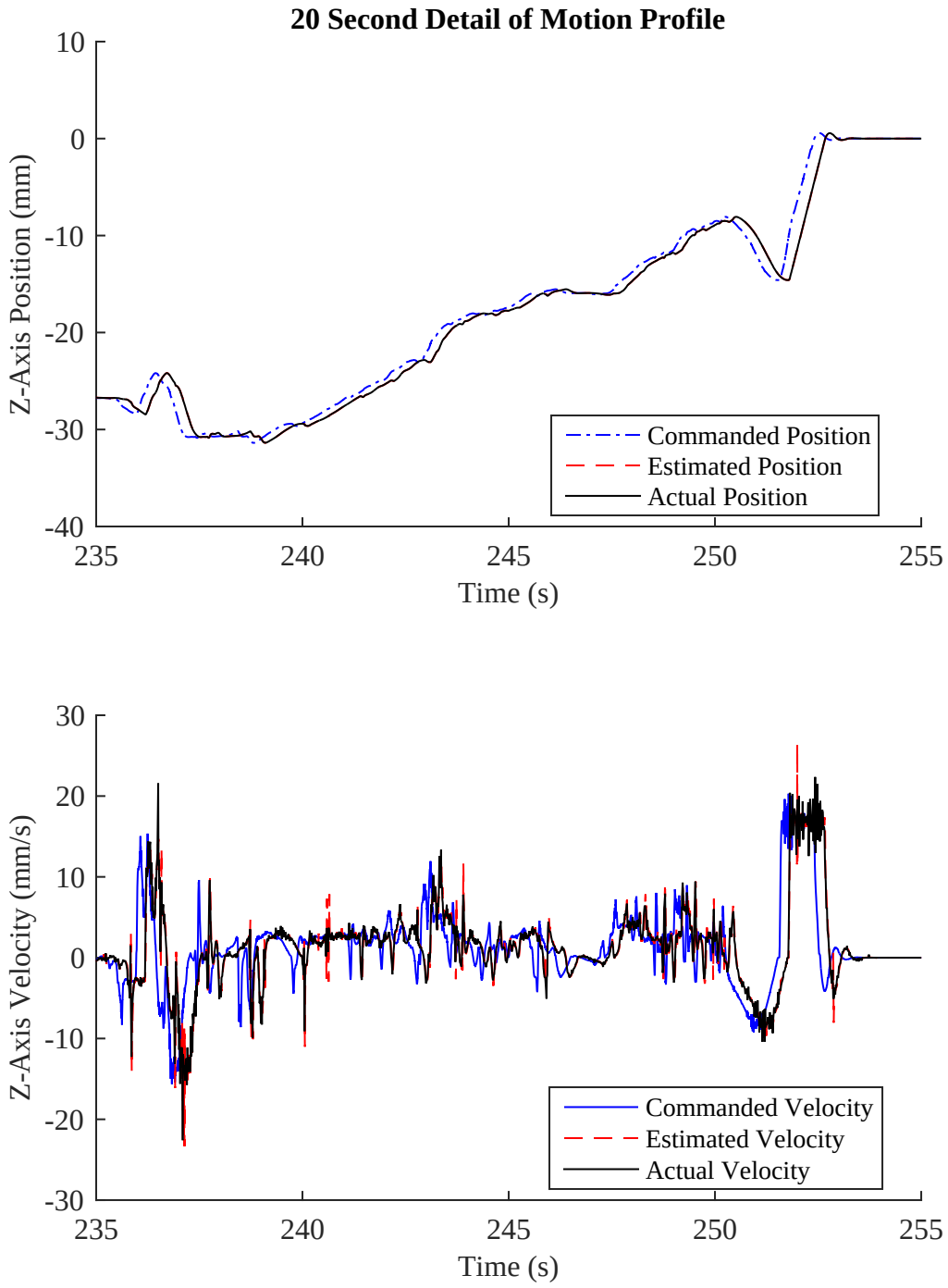


Figure 4.30: 20 Second Z-axis Position and Velocity Progression for Head Top Toolpath

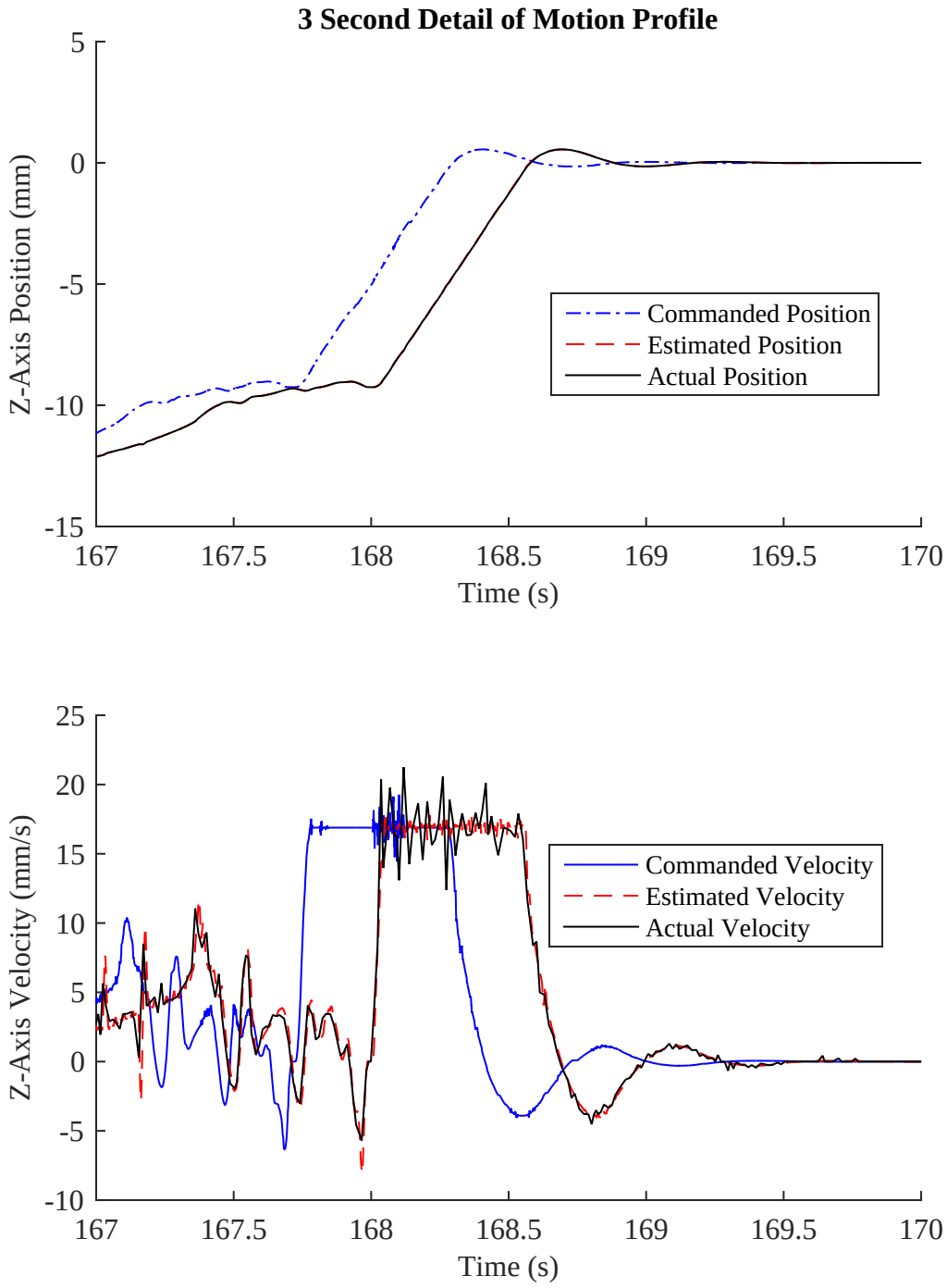


Figure 4.31: 3 Second Z-axis Position and Velocity Progression for Head Top Toolpath

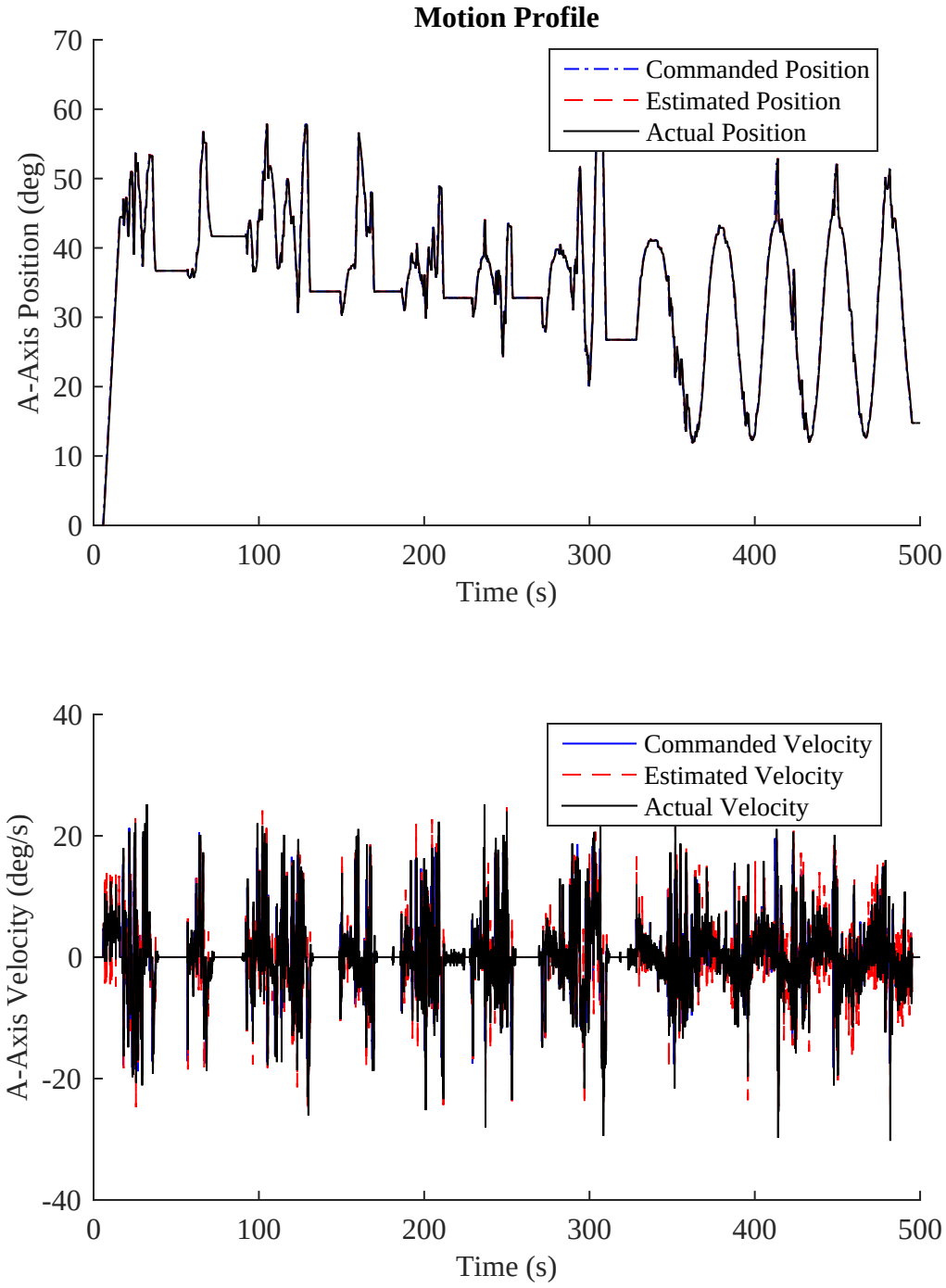


Figure 4.32: A-axis Position and Velocity Progression for Head Top Toolpath

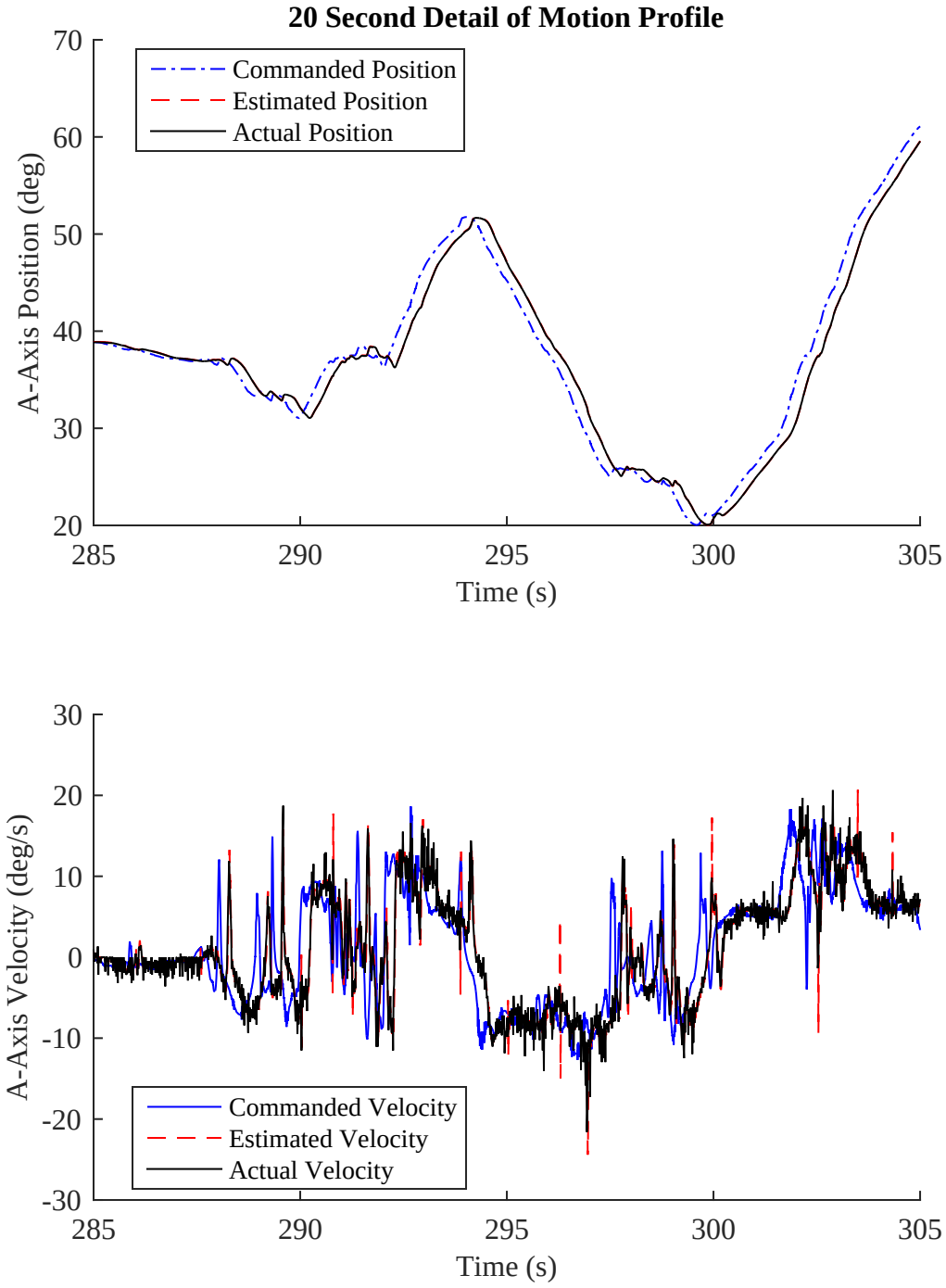


Figure 4.33: 20 Second A-axis Position and Velocity Progression for Head Top Toolpath

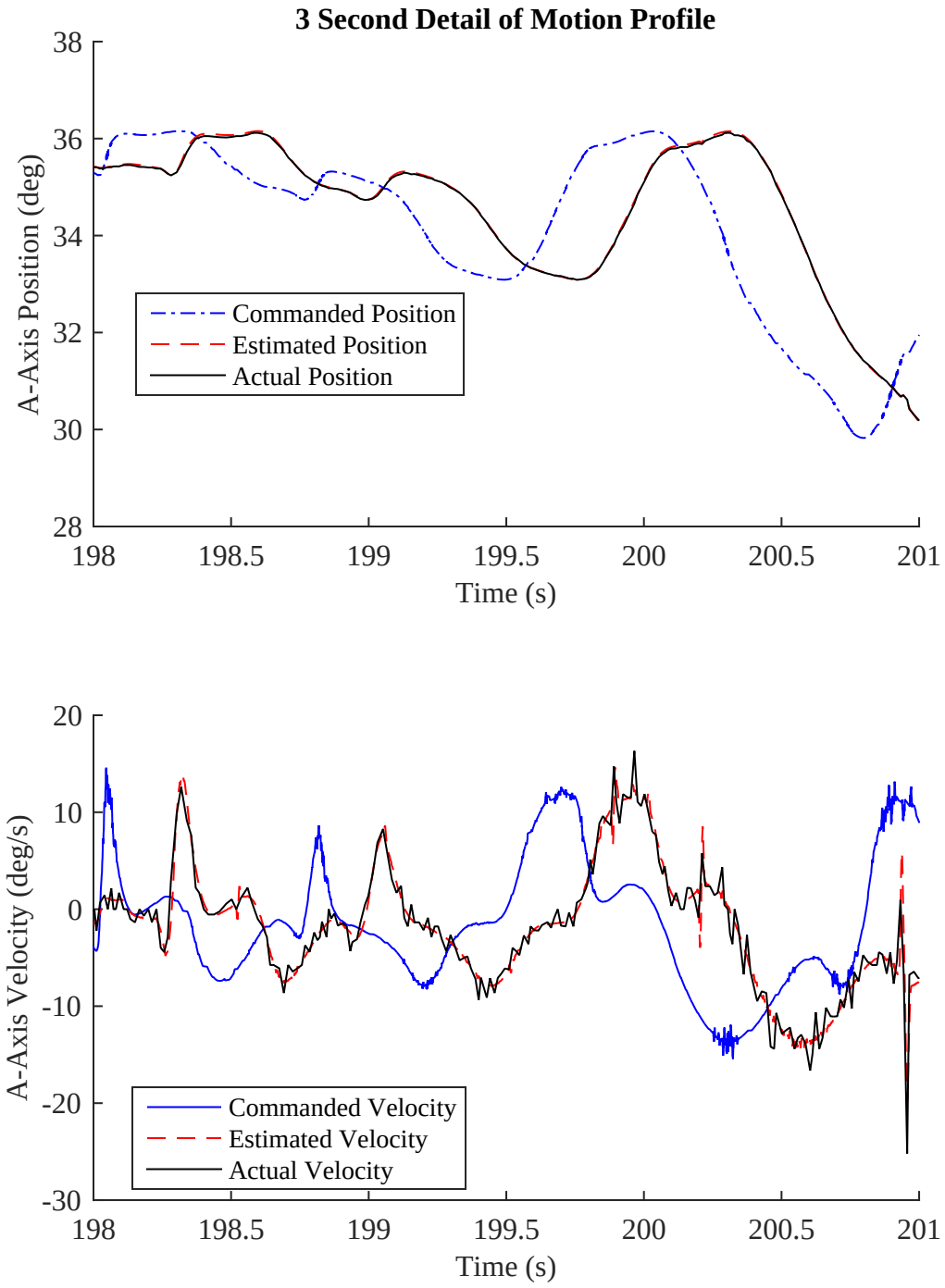


Figure 4.34: 3 Second A-axis Position and Velocity Progression for Head Top Toolpath

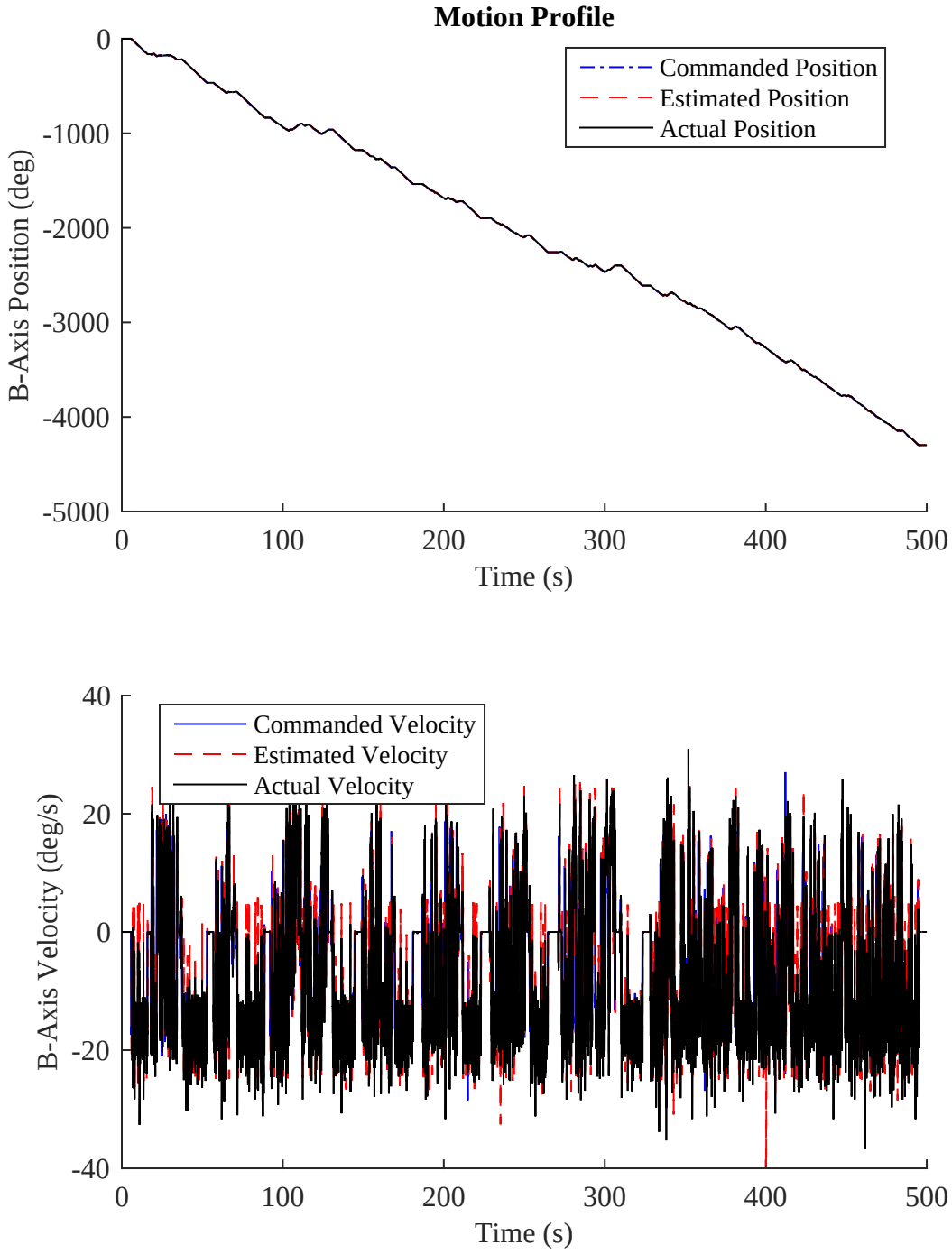


Figure 4.35: B-axis Position and Velocity Progression for Head Top Toolpath

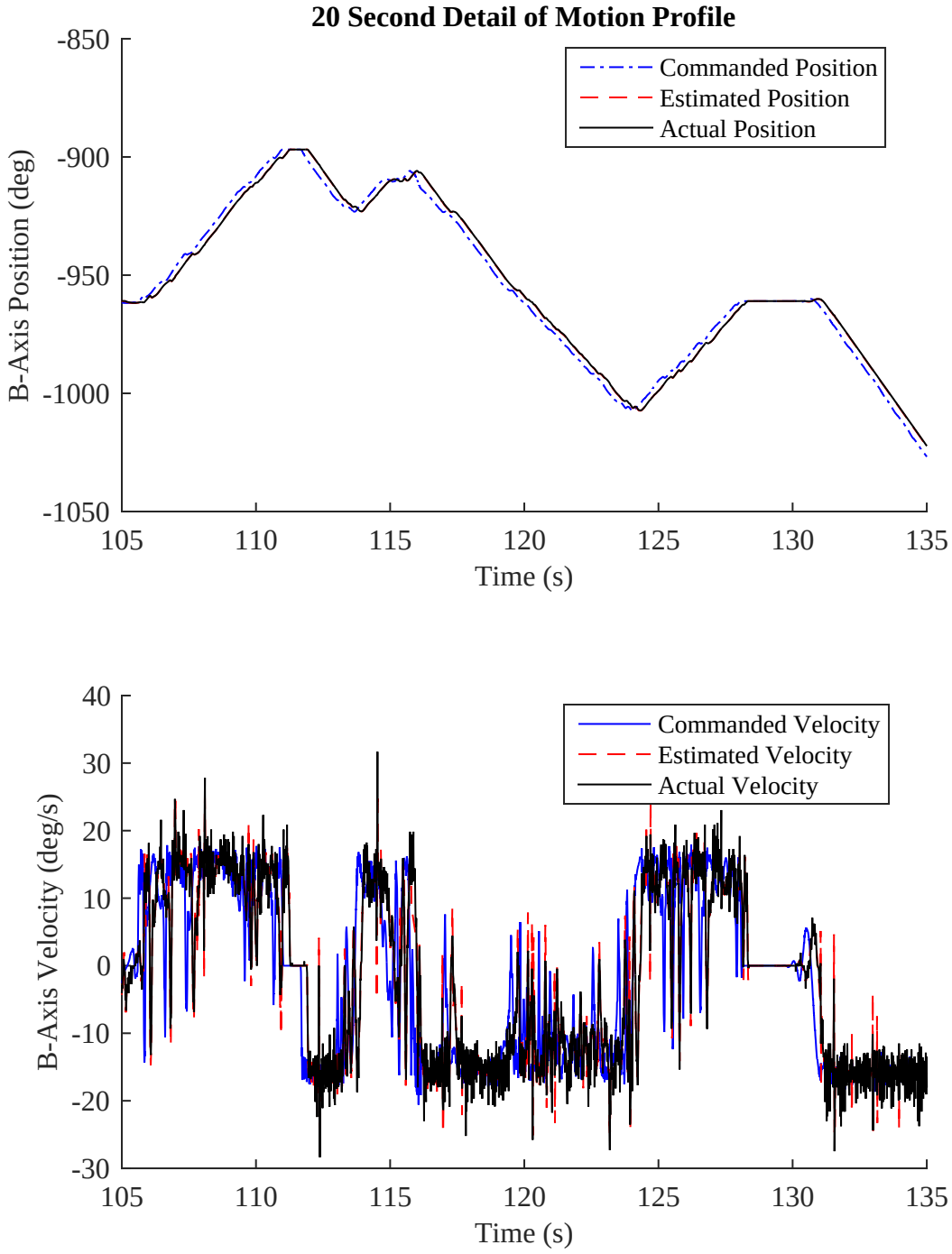


Figure 4.36: 20 Second B-axis Position and Velocity Progression for Head Top Toolpath

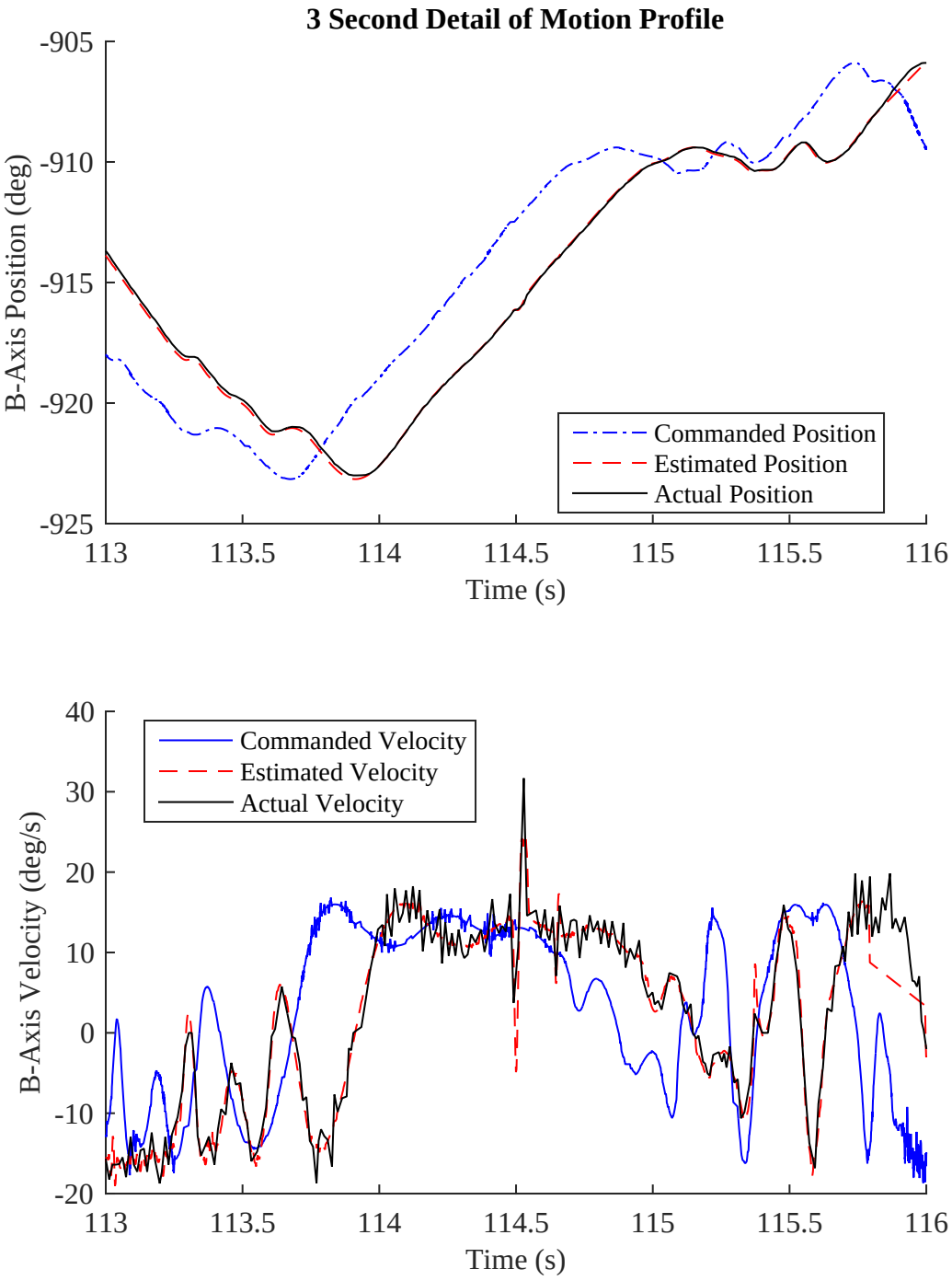


Figure 4.37: 3 Second B-axis Position and Velocity Progression for Head Top Toolpath

Tool Space Data

A 3-dimensional visualization of the entire toolpath is presented in Figure 4.38, where the fifty largest trajectory errors (as measured by axis encoders) are shown with green circles. The commanded toolpath is shown as a blue solid line, the estimated toolpath is shown as a red dashed line, and the actual toolpath measured by the encoders is shown as a black solid line. A detail view of the five largest trajectory errors during execution (the largest of which was 249.2μm, as reported in Table 4.4) is shown in Figure 4.39.

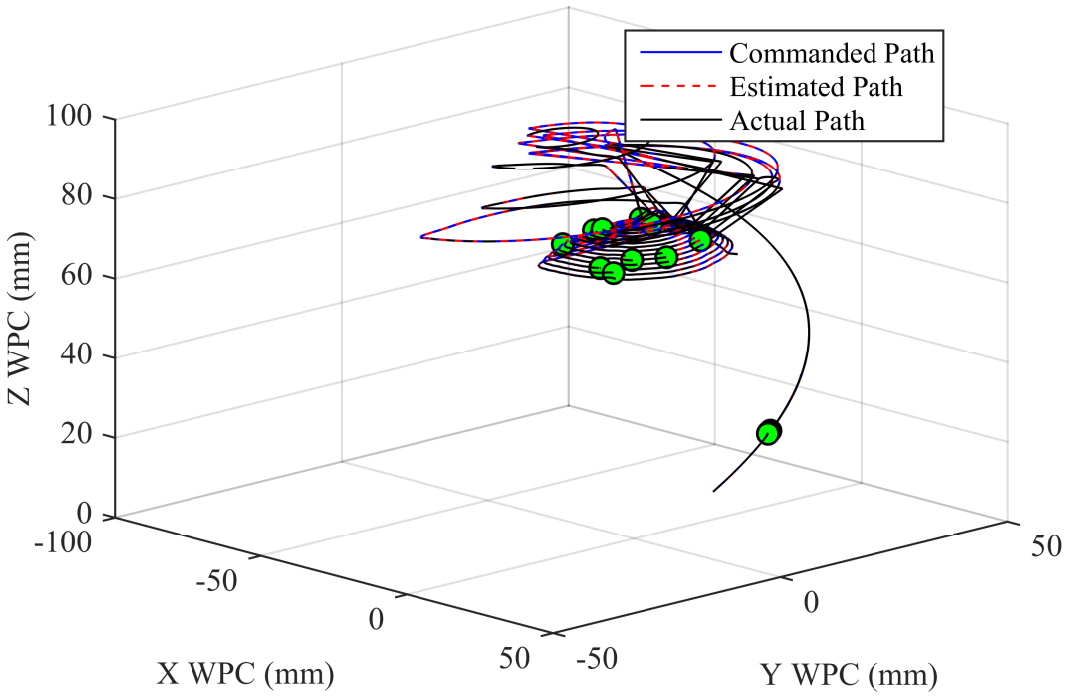


Figure 4.38: 3-Dimensional Visualization of Head Top Toolpath

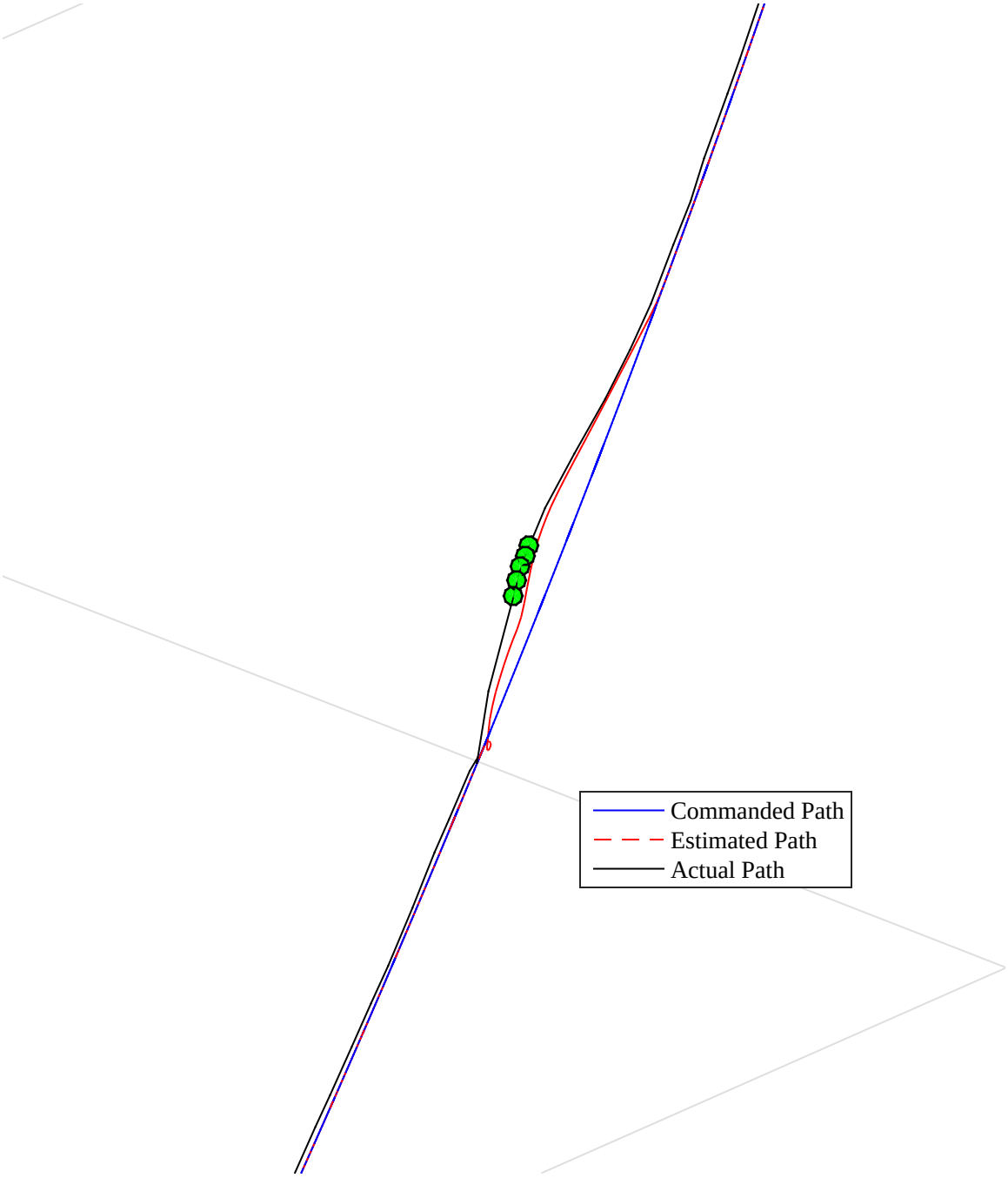


Figure 4.39: Detail of Largest Trajectory Error in Head Top Toolpath

4.4.3 Head Bottom Toolpath

Joint Space Data

Similar experimental results were obtained for the toolpath that finishes the bottom of the head model. For brevity, the complete axis position traces are shown in Figures A.1, A.3, A.5, A.7, and A.9 in Appendix A. Additionally, the twenty second detail views are shown in Figures A.2, A.4, A.6, A.8, and A.10 in Appendix A. However, the three second detail views in Figures 4.40, 4.41, 4.42, 4.43, 4.44 are shown in this chapter for comparison with the other experimental toolpaths.

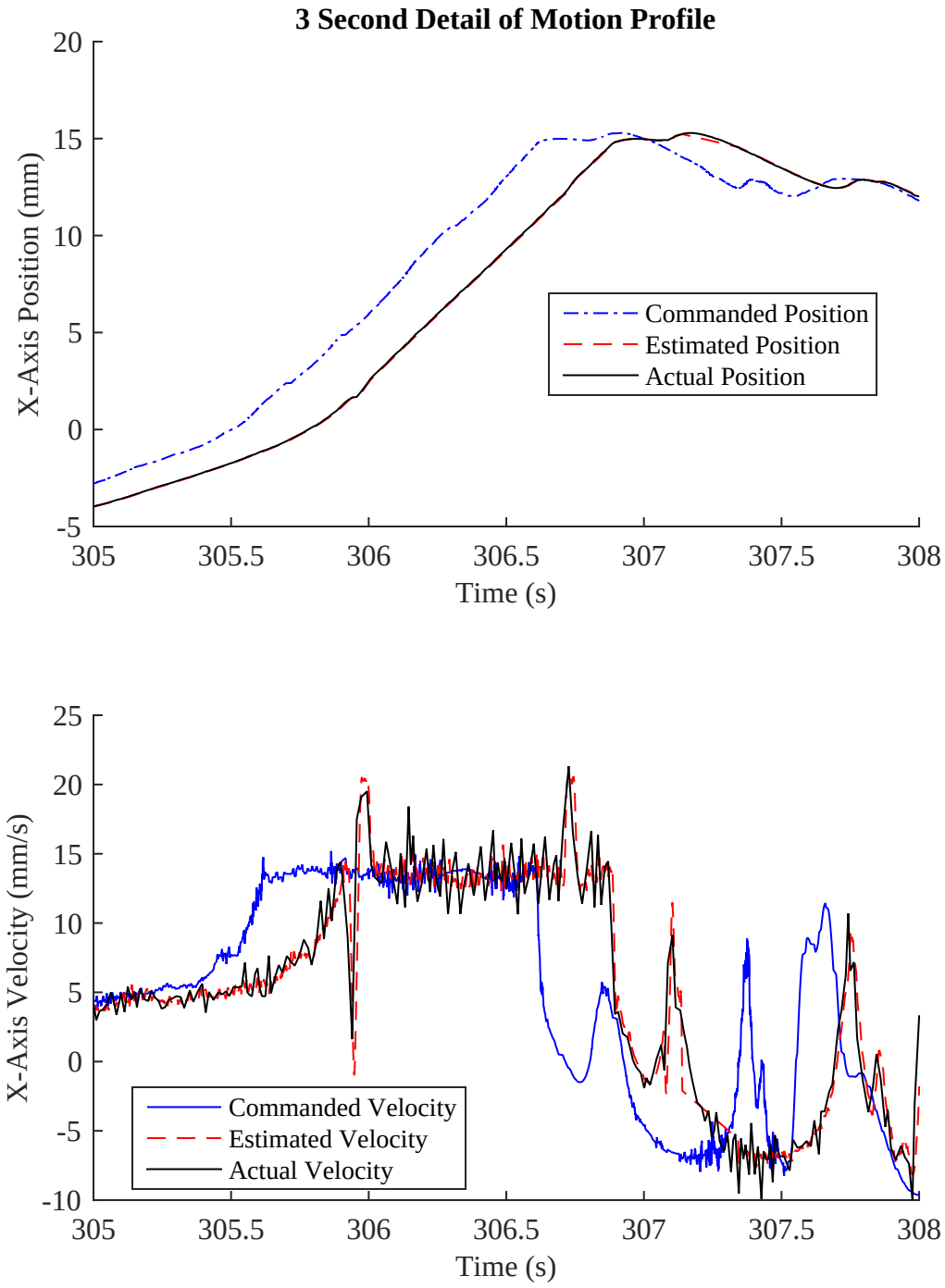


Figure 4.40: 3 Second X-axis Position and Velocity Progression for Head Bottom Toolpath

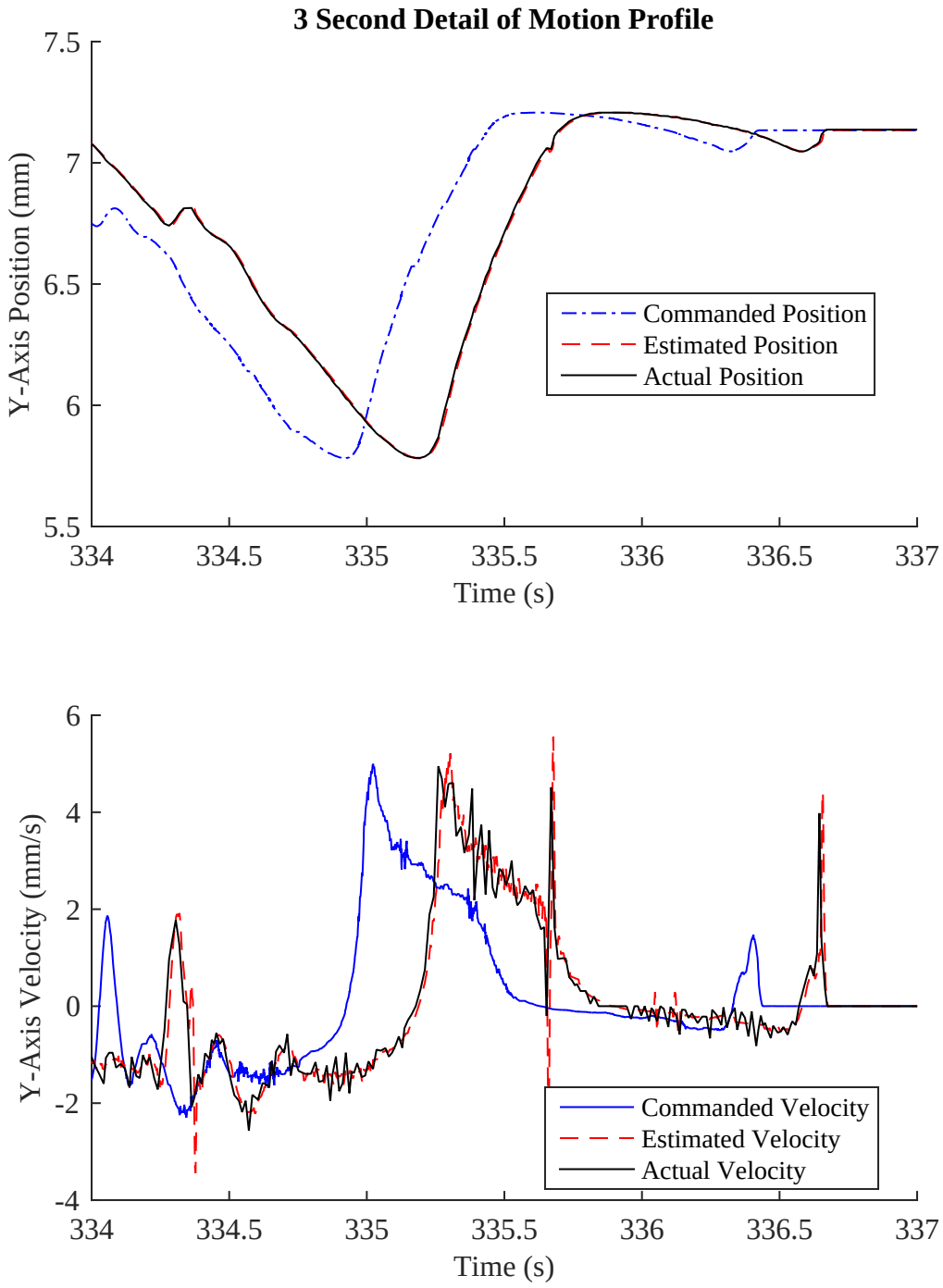


Figure 4.41: 3 Second Y-axis Position and Velocity Progression for Head Bottom Toolpath

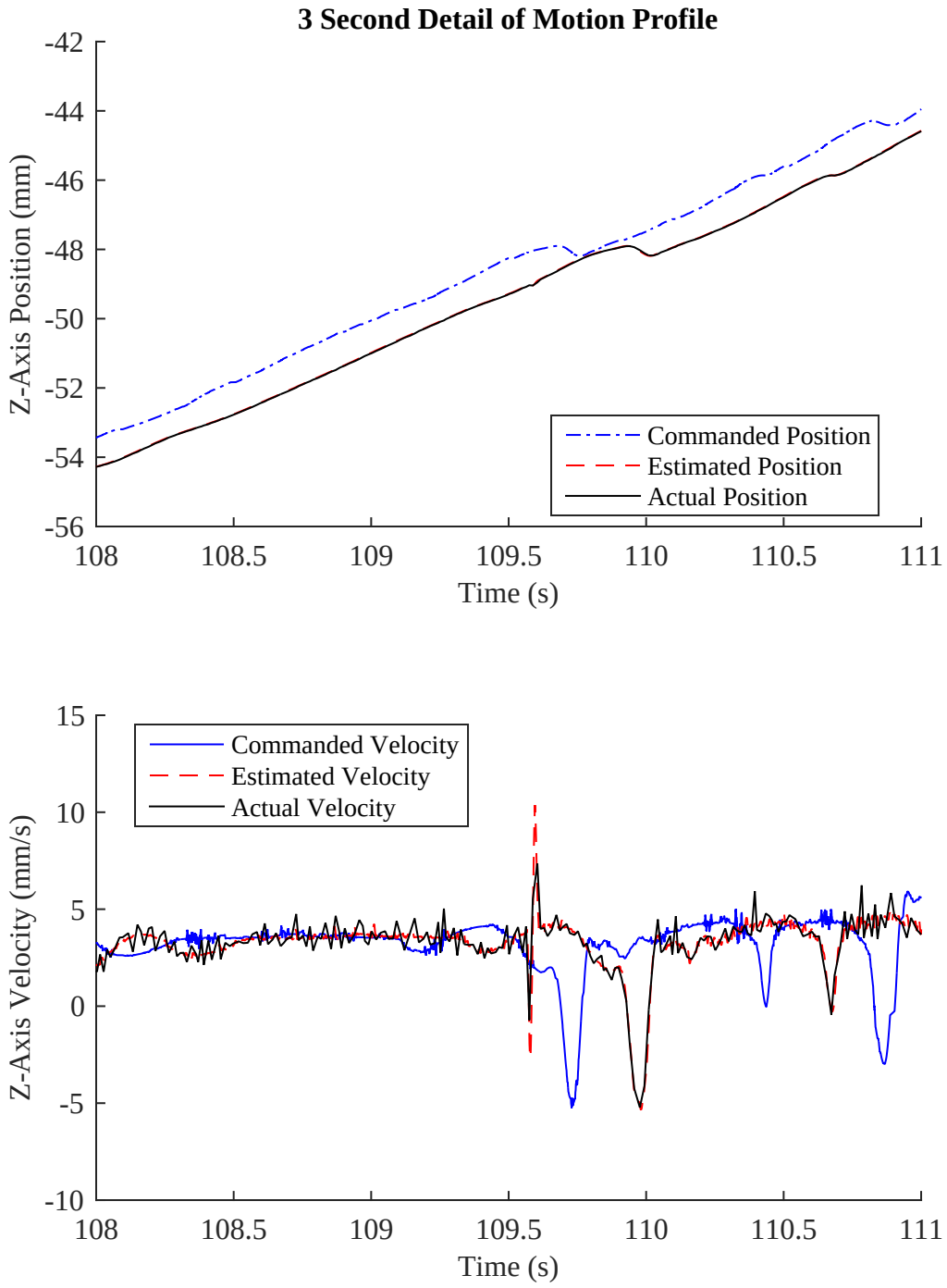


Figure 4.42: 3 Second Z-axis Position and Velocity Progression for Head Bottom Toolpath

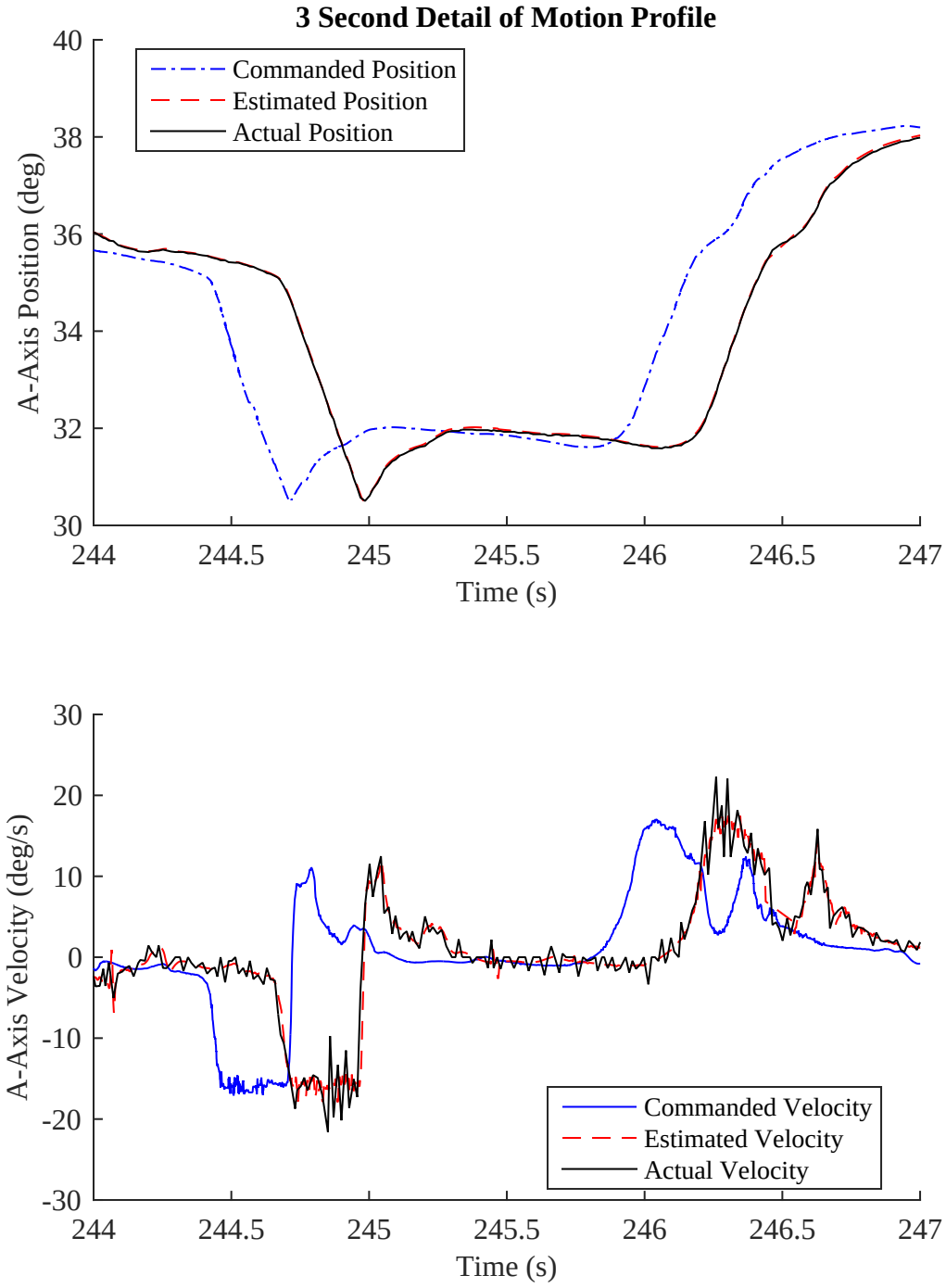


Figure 4.43: 3 Second A-axis Position and Velocity Progression for Head Bottom Toolpath

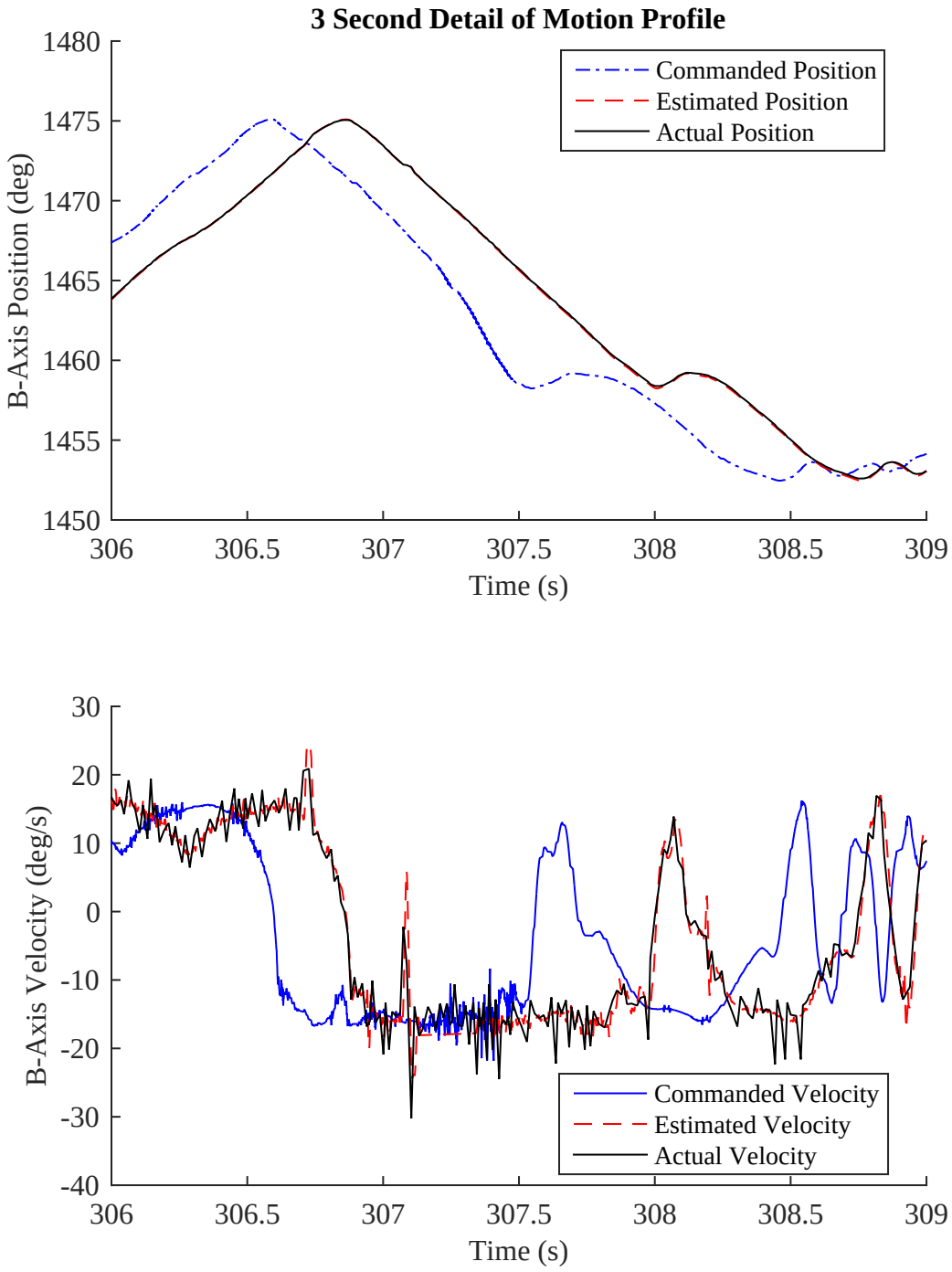


Figure 4.44: 3 Second B-axis Position and Velocity Progression for Head Bottom Toolpath

Tool Space Data

A 3-dimensional visualization of the entire toolpath used to finish the bottom of the head model, in addition to the fifty largest trajectory errors is presented in Figure 4.45. The commanded toolpath is shown as a blue solid line, the estimated toolpath is shown as a red dashed line, and the actual toolpath measured by the encoders is shown as a black solid line. A detail view of the five largest trajectory errors during execution (the largest of which was 232.9μm, as reported in Table 4.4) is shown in Figure 4.46.

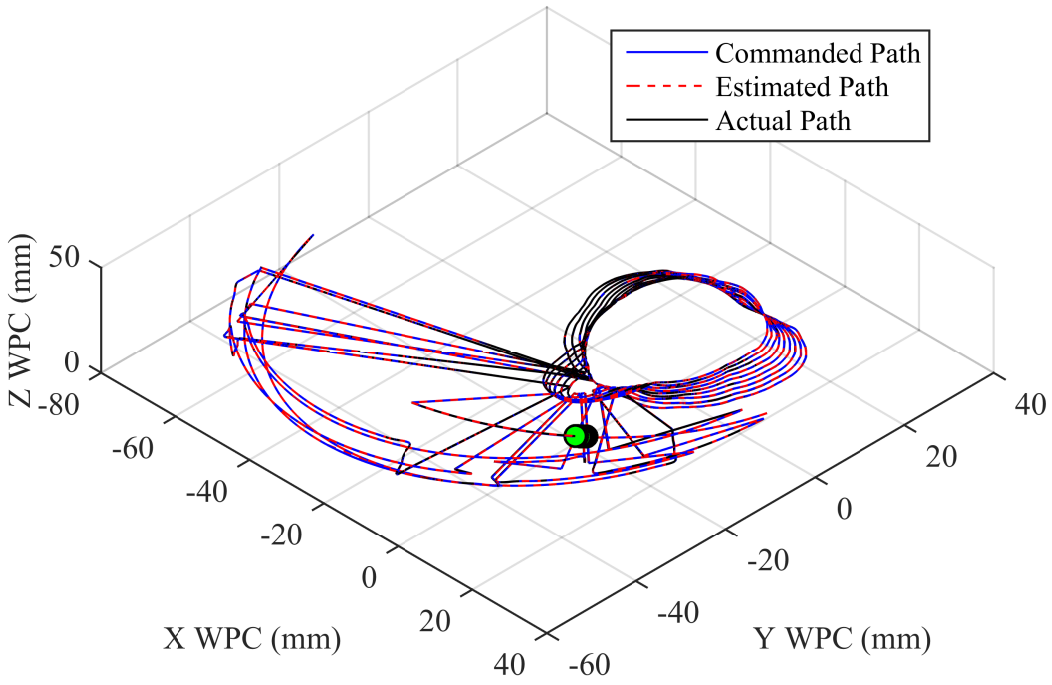


Figure 4.45: 3-Dimensional Visualization of Head Bottom Toolpath

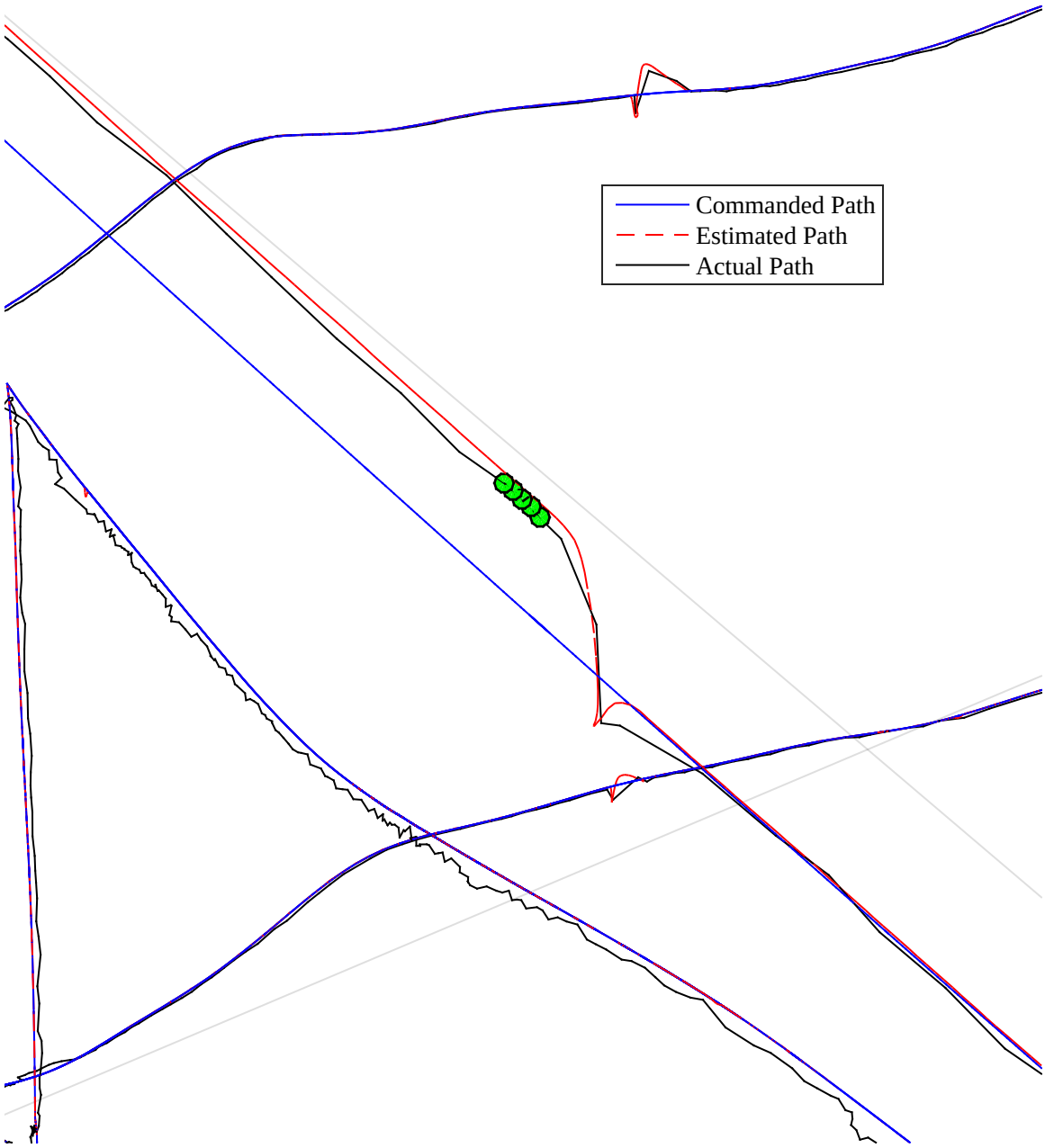


Figure 4.46: Detail of Largest Trajectory Error in Head Bottom Toolpath

4.4.4 Candleholder Top Toolpath

Joint Space Data

Figures A.11, A.13, A.15, A.17, and A.19 in Appendix A show position and velocity progressions for the entire toolpath used to finish the top of the candleholder model; Figures A.12, A.14, A.16, A.18, and A.20 in Appendix A show twenty second detail views of the progressions; and Figures 4.47, A.14, A.16, A.18, and A.20 show three second detail views.

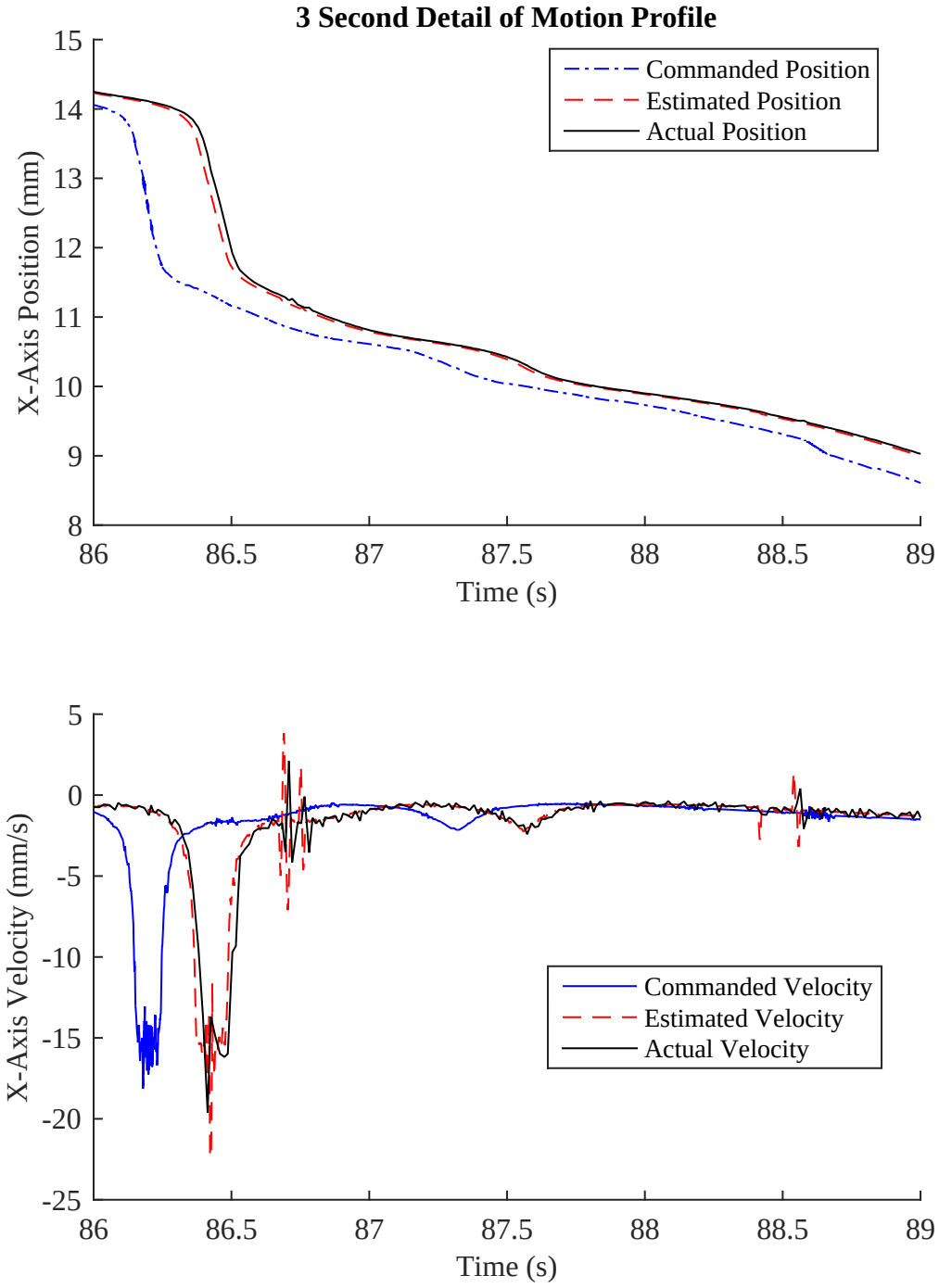


Figure 4.47: 3 Second X-axis Position and Velocity Progression for Candleholder Top Toolpath

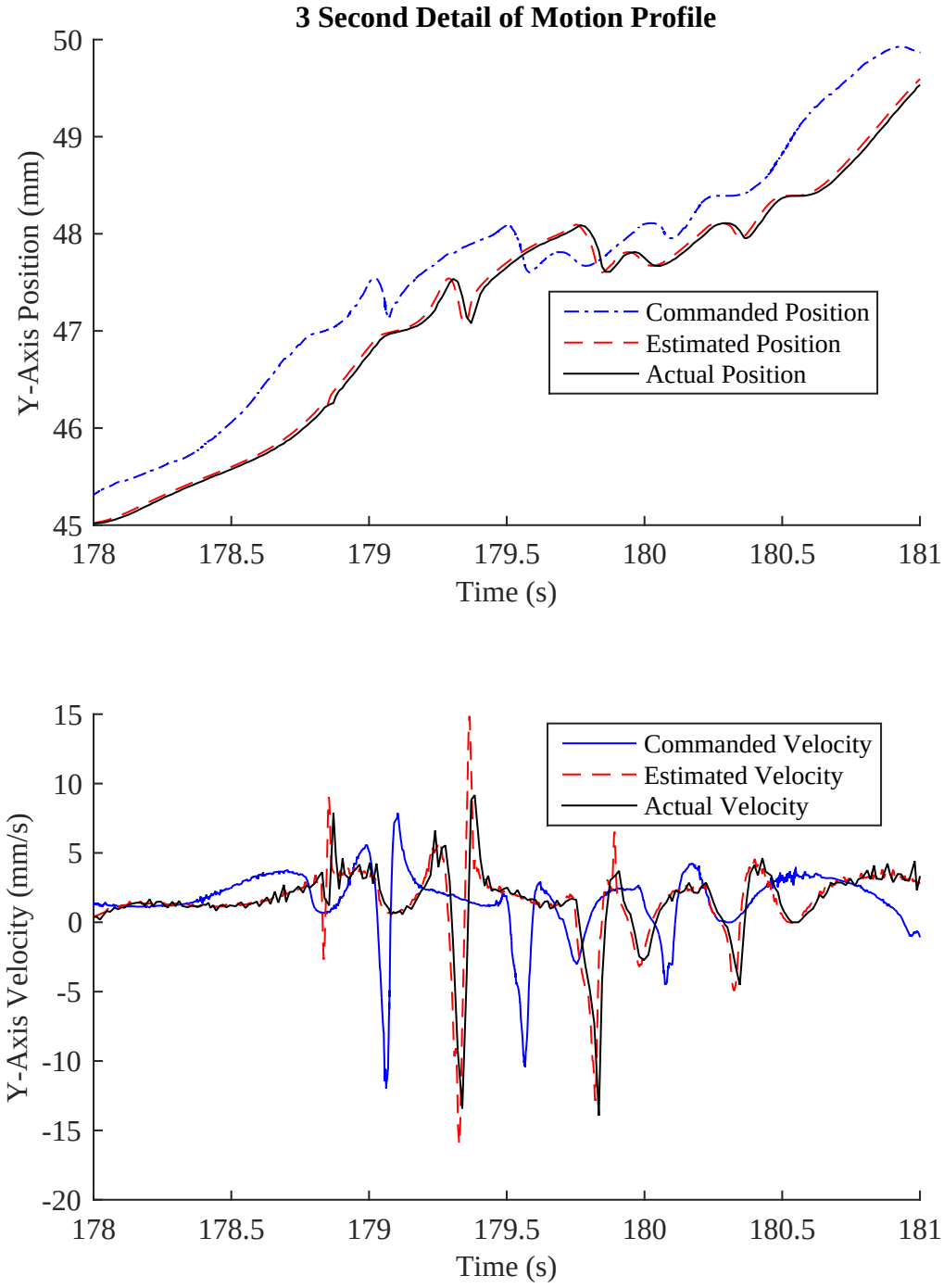


Figure 4.48: 3 Second Y-axis Position and Velocity Progression for Candleholder Top Toolpath

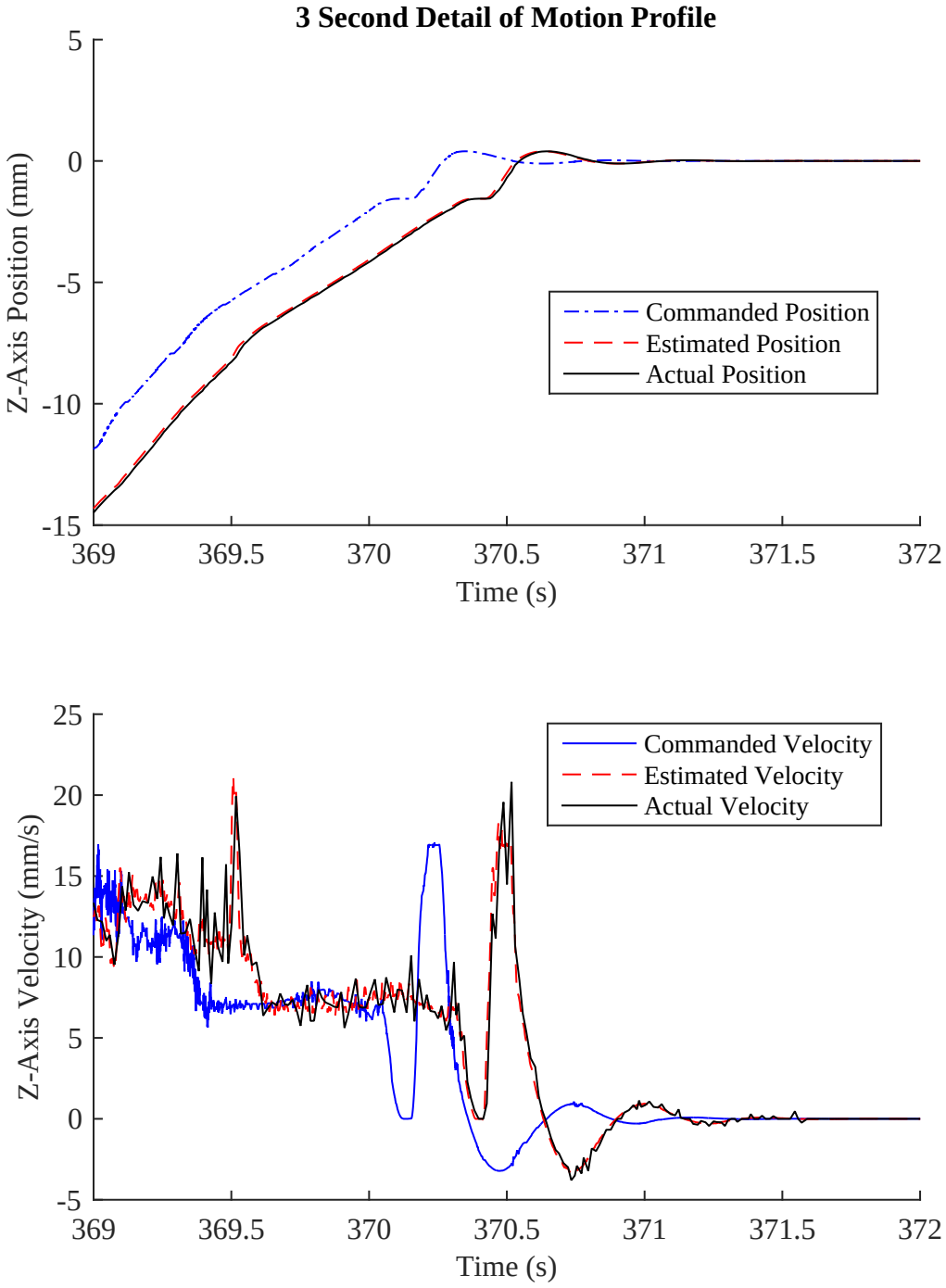


Figure 4.49: 3 Second Z-axis Position and Velocity Progression for Candleholder Top Toolpath

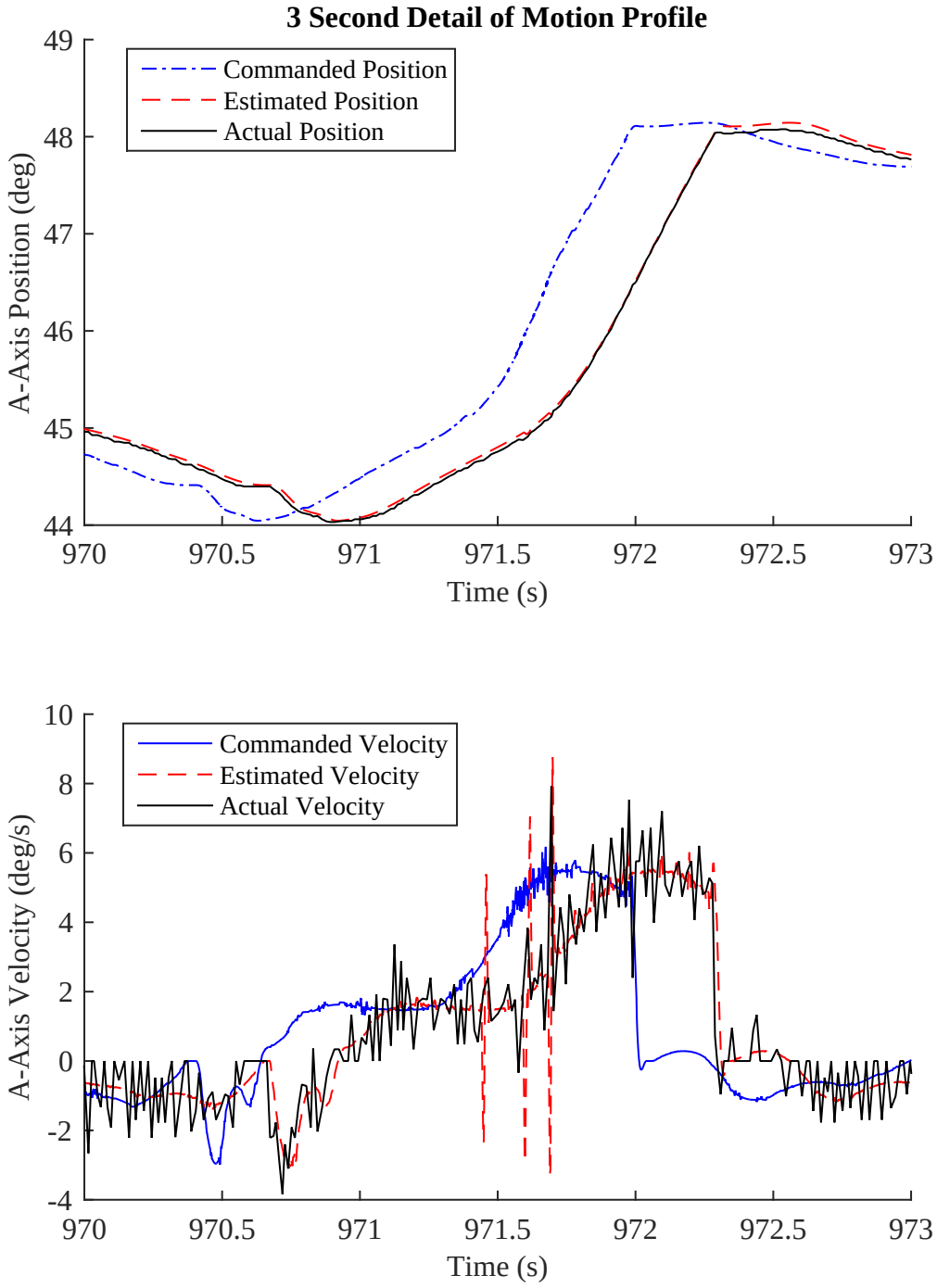


Figure 4.50: 3 Second A-axis Position and Velocity Progression for Candleholder Top Toolpath

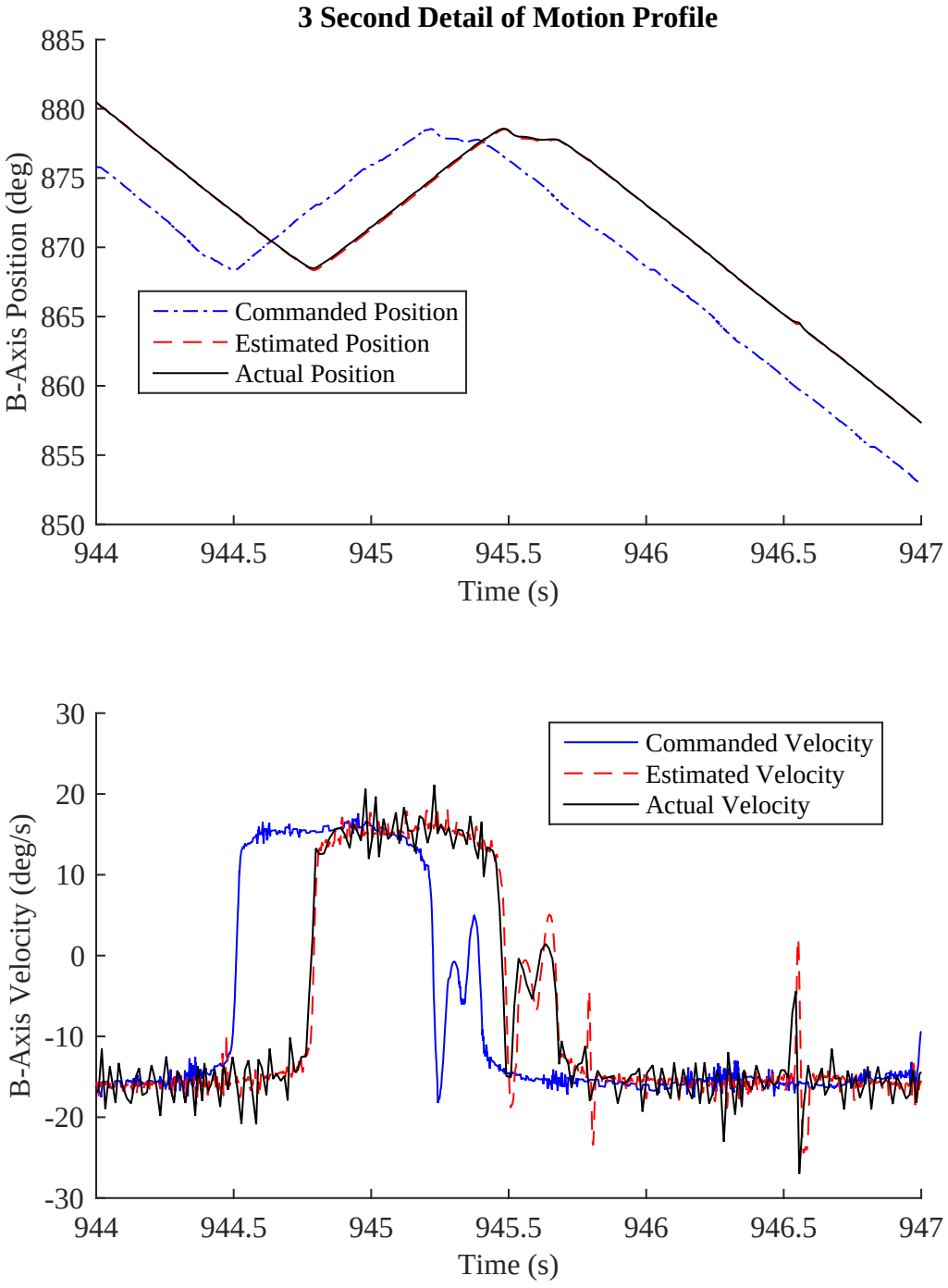


Figure 4.51: 3 Second B-axis Position and Velocity Progression for Candleholder Top Toolpath

Tool Space Data

A 3-dimensional visualization of the entire toolpath used to finish the top of the candleholder model, in addition to the fifty largest trajectory errors is presented in Figure 4.52. The commanded toolpath is shown as a blue solid line, the estimated toolpath is shown as a red dashed line, and the actual toolpath measured by the encoders is shown as a black solid line. A detail view of the five largest trajectory errors during execution (the largest of which was 367.8μm, as reported in Table 4.4) is shown in Figure 4.53.

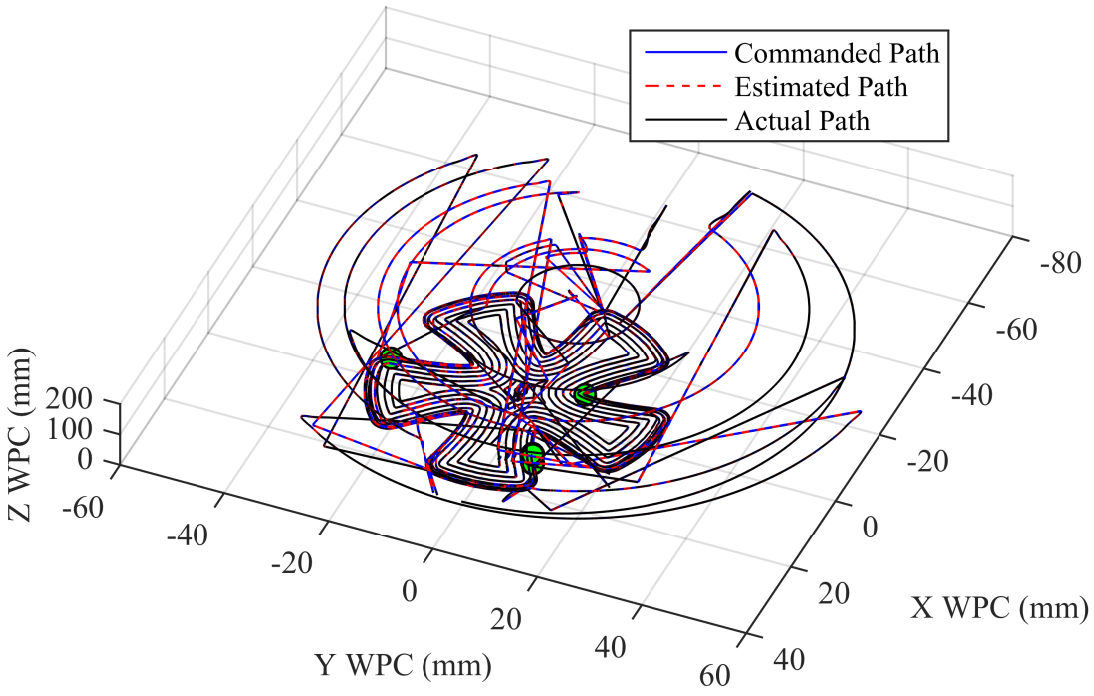


Figure 4.52: 3-Dimensional Visualization of Candleholder Top Toolpath

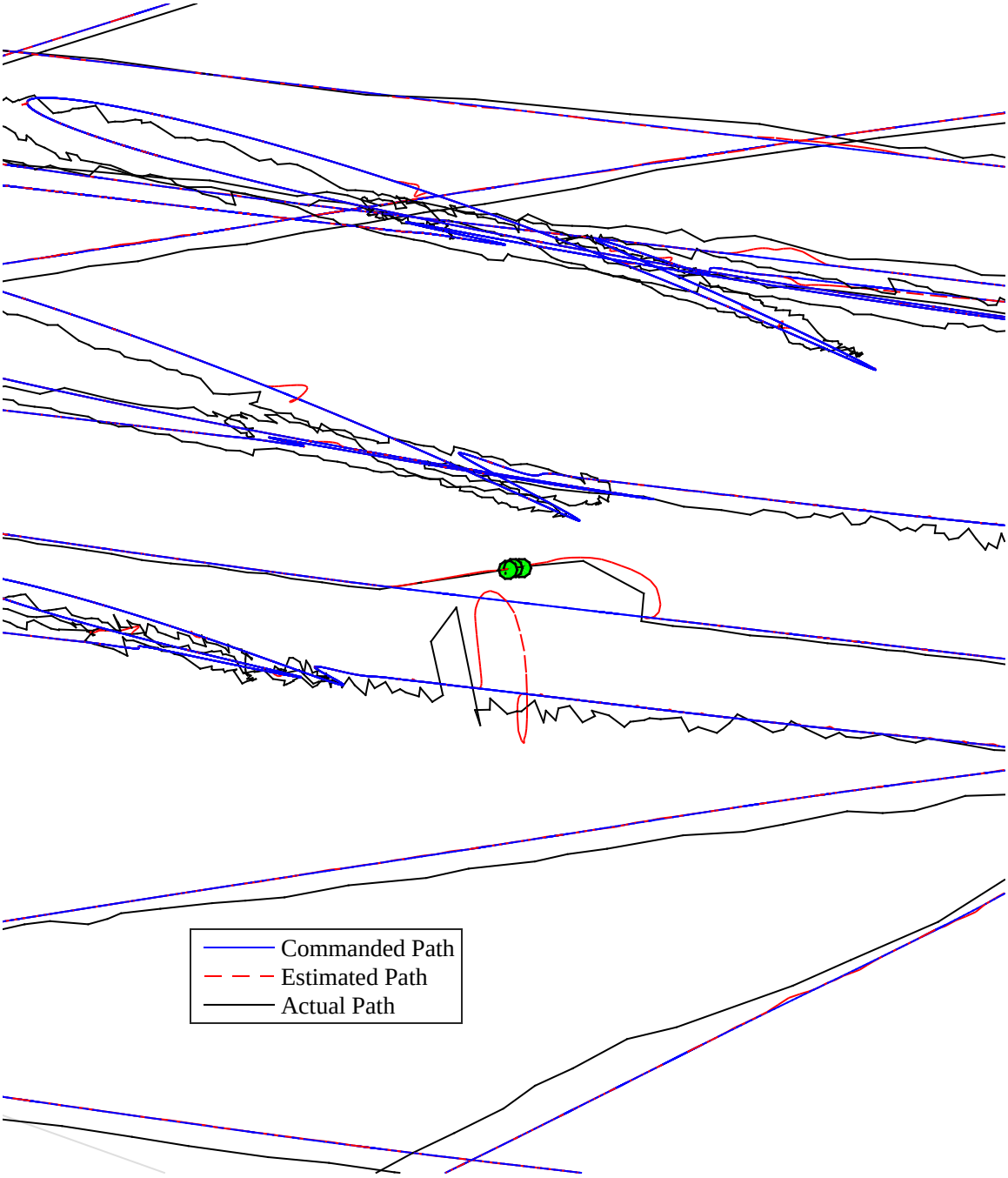


Figure 4.53: Detail of Largest Trajectory Error in Candleholder Top Toolpath

4.4.5 Candleholder Bottom Toolpath

Joint Space Data

The final pass used to finish a portion of the bottom of the candleholder model is the shortest of the four experimental paths, and full position and velocity progressions are shown in Figures A.21, A.23 A.25, A.27, and A.29 in Appendix A; twenty second detail views are shown in Figures A.22, A.24, A.26, A.28, and A.30 in Appendix A; and three second detail views are shown in Figures 4.54, 4.55, 4.56, 4.57, and 4.58. As expected, these results are similar to the previous three experimental toolpaths.

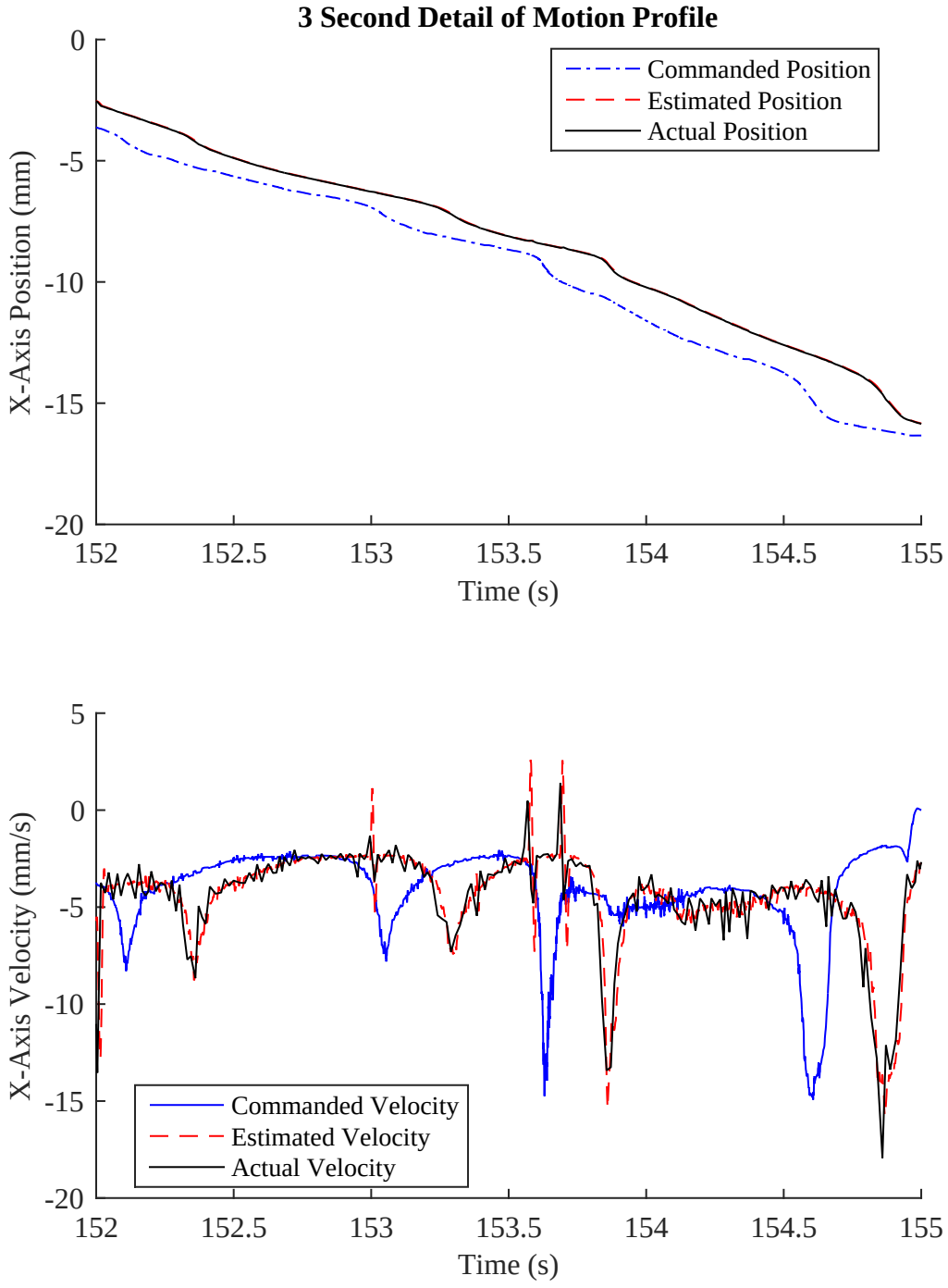


Figure 4.54: 3 Second X-axis Position and Velocity Progression for Candleholder Bottom Toolpath

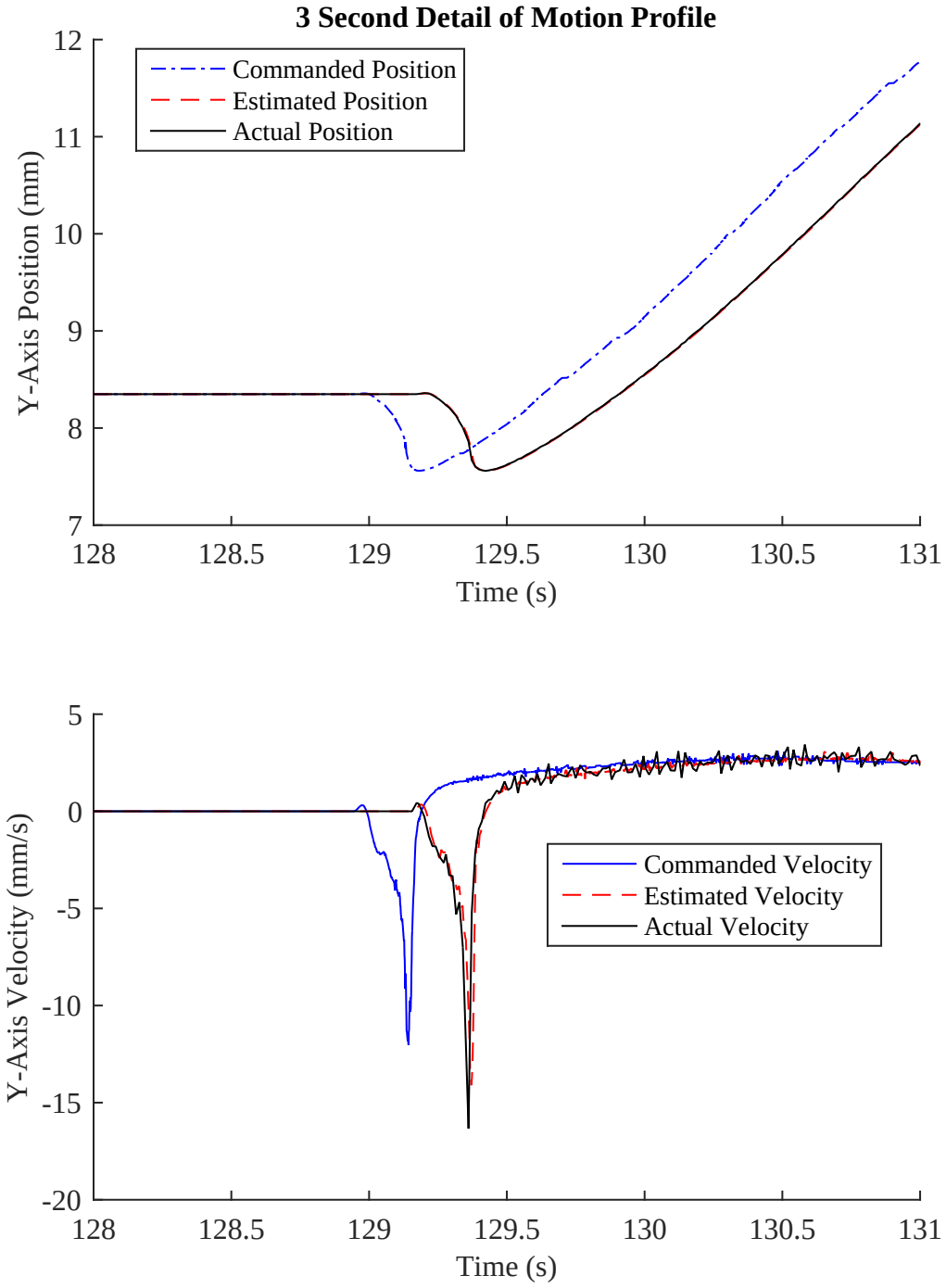


Figure 4.55: 3 Second Y-axis Position and Velocity Progression for Candleholder Bottom Toolpath

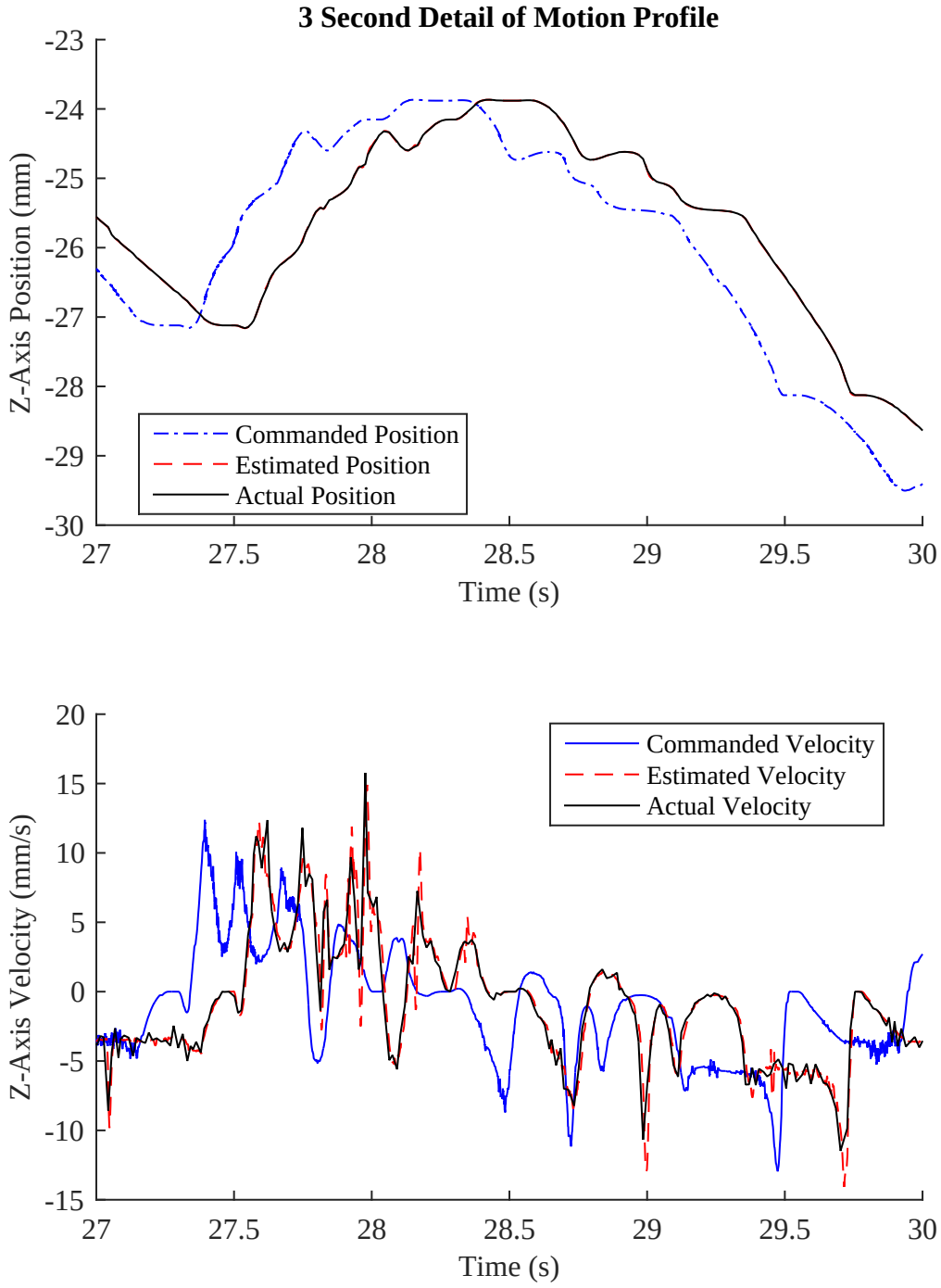


Figure 4.56: 3 Second Z-axis Position and Velocity Progression for Candleholder Bottom Toolpath

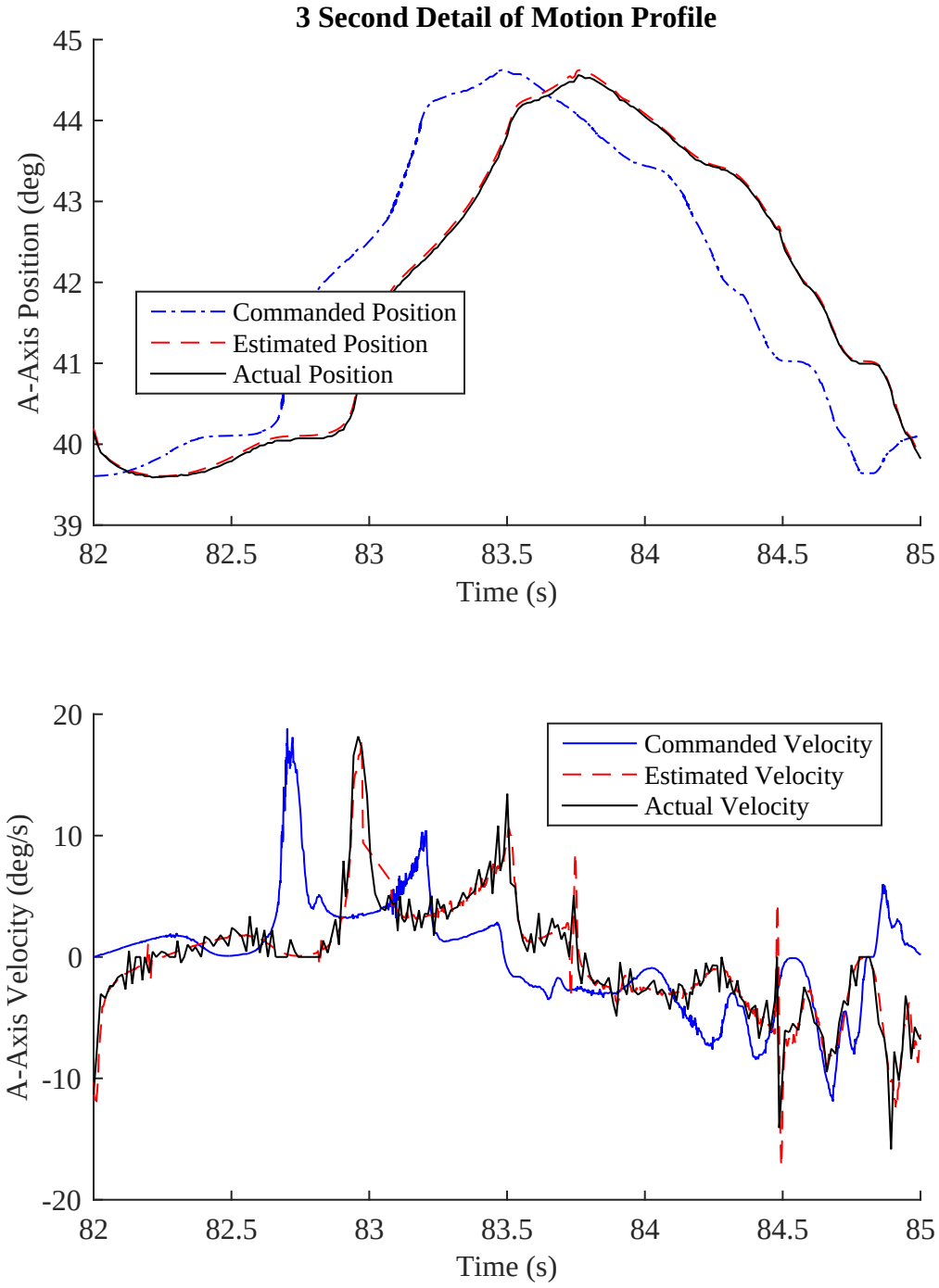


Figure 4.57: 3 Second A-axis Position and Velocity Progression for Candleholder Bottom Toolpath

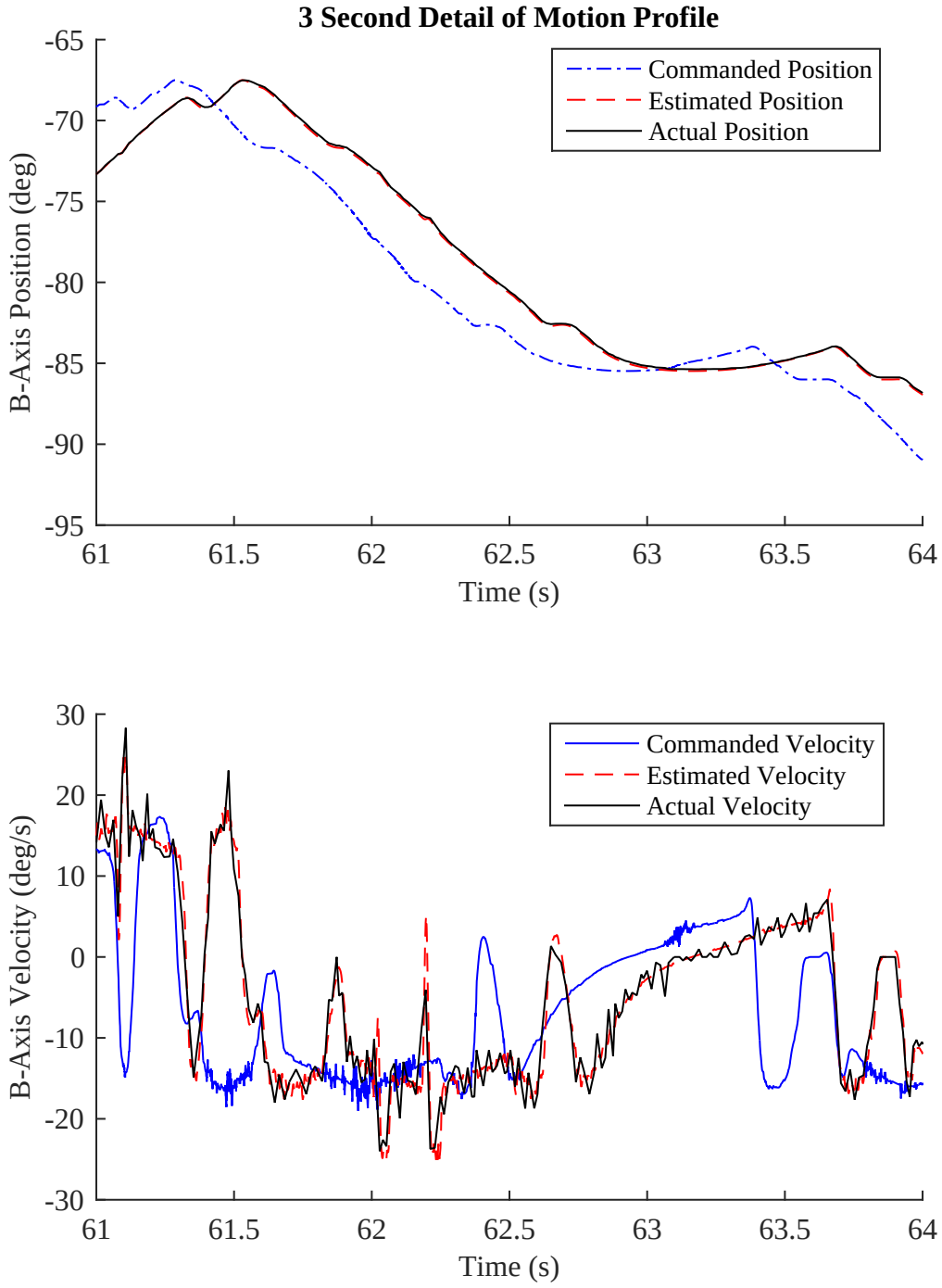


Figure 4.58: 3 Second B-axis Position and Velocity Progression for Candleholder Bottom Toolpath

Tool Space Data

A 3-dimensional visualization of the entire toolpath used to finish the bottom of the candleholder model, in addition to the fifty largest trajectory errors is presented in Figure 4.59. The commanded toolpath is shown as a blue solid line, the estimated toolpath is shown as a red dashed line, and the actual toolpath measured by the encoders is shown as a black solid line. A detail view of the five largest trajectory errors during execution (the largest of which was 211.1μm, as reported in Table 4.4) is shown in Figure 4.60.

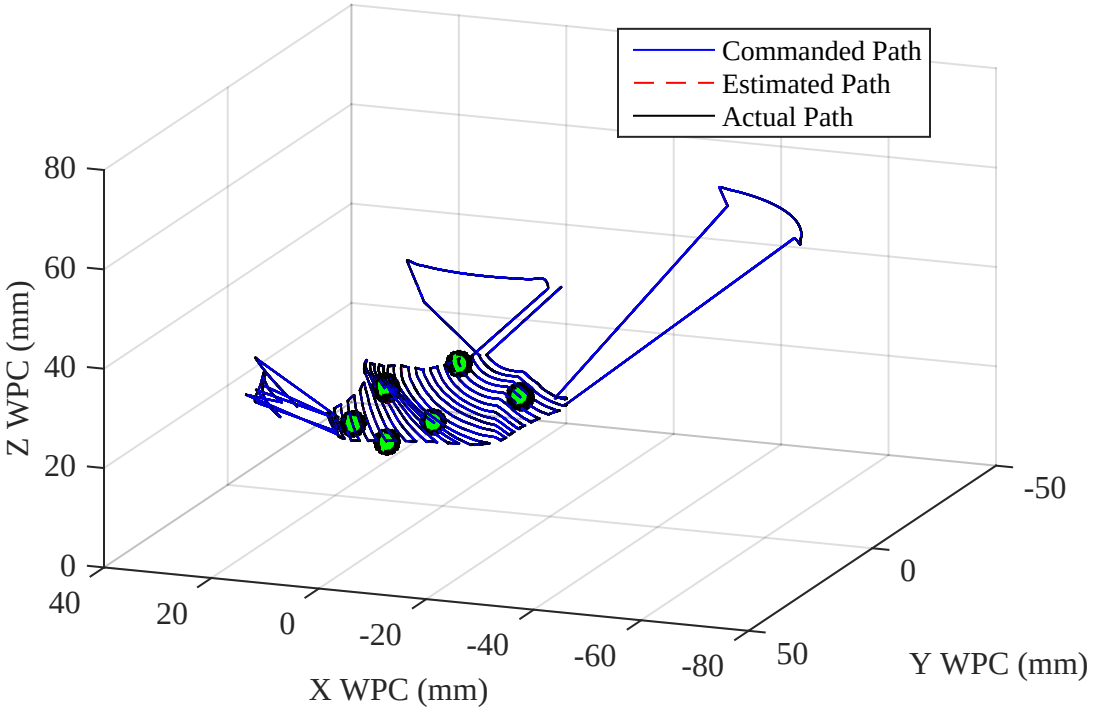


Figure 4.59: 3-Dimensional Visualization of Candleholder Bottom Toolpath

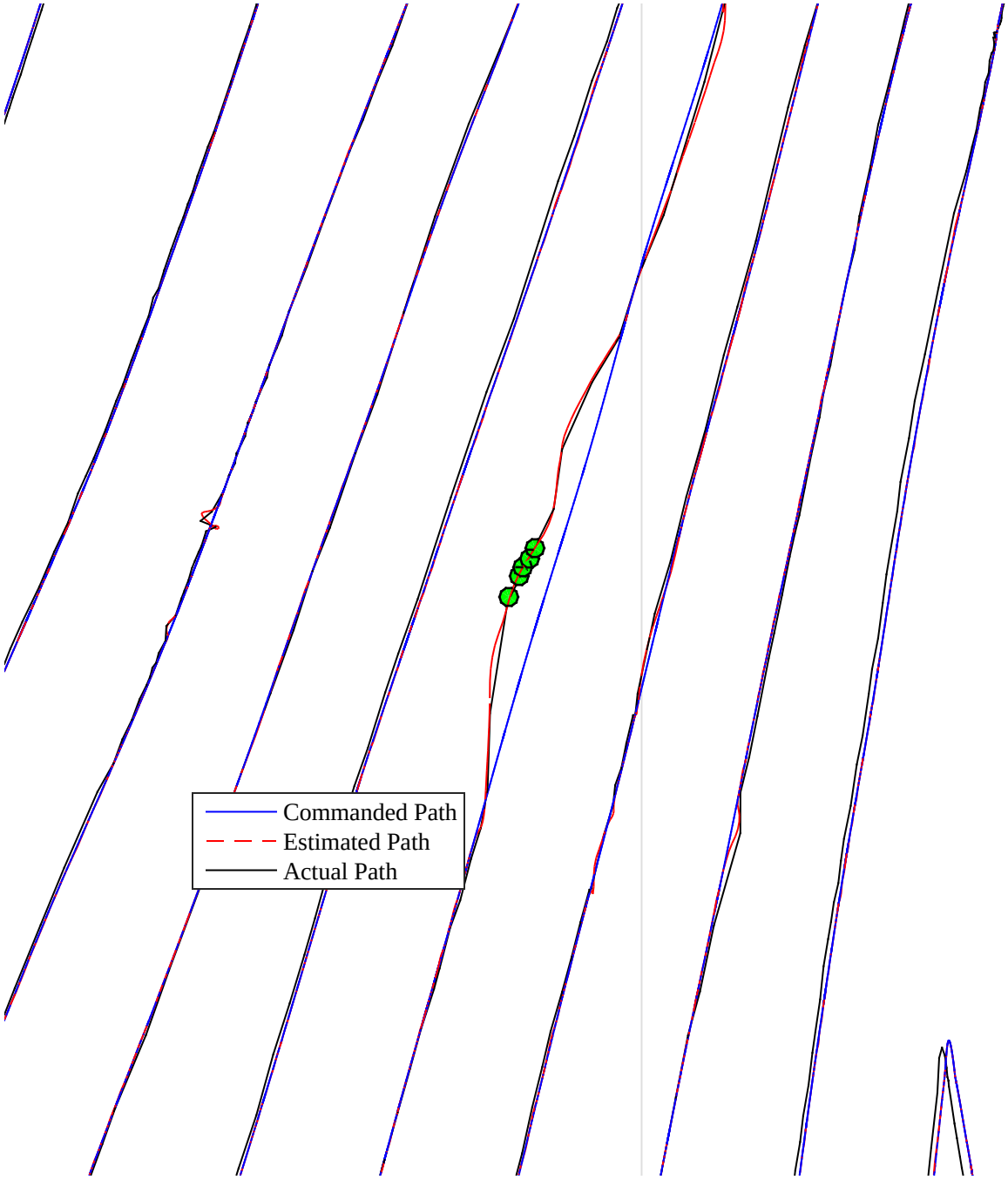


Figure 4.60: Detail of Largest Trajectory Error in Candleholder Bottom Toolpath

4.5 Comparison with G-Code Control

The toolpaths that were executed using the direct control system were compared to the same toolpaths executed using G-Code to determine differences in machining time. These experiments demonstrate the benefits of offline trajectory planning, as axis velocity saturation is not experienced due to finite lookahead buffer length. The results presented in Table 4.3 are reproduced in Table 4.5, along with the corresponding execution time for the same G-Code trajectory; additionally, the reduction in machining time for the directly controlled toolpath as compared to the G-Code toolpath is presented in the last column for each toolpath. It is apparent from these results that the direct servo control strategy developed in this work enables faster toolpath traversal due to both time optimal trajectory planning and the elimination of the lookahead buffer.

Toolpath	**G-Code Execution Time (s)**	**Directly Controlled Execution Time (s)**	**Reduction**
Head Top	534.365	492.471	7.84%
Head Bottom	525.995	465.630	11.48%
Candleholder Top	1304.874	1042.701	20.09%
Candleholder Bottom	216.973	158.788	26.83%

Table 4.5: Execution Time Comparison for Experimental Toolpaths

The differences in execution time gains between the toolpaths can be explained by examining some characteristics of each of the paths. Each path has a different total length, different motion profiles, and different numbers of retractions, which makes direct comparison of the performance gains difficult. It is expected that the time-optimal trajectory planning strategy will be more effective in regions where multiple simultaneous axes are executing complex motion profiles (which is frequently the case in cutting regions), since trapezoidal velocity profile trajectory planning produces time optimal results for simple motion, such as retractions.

More insight into the performance gains observed in the experimental data can be ob-

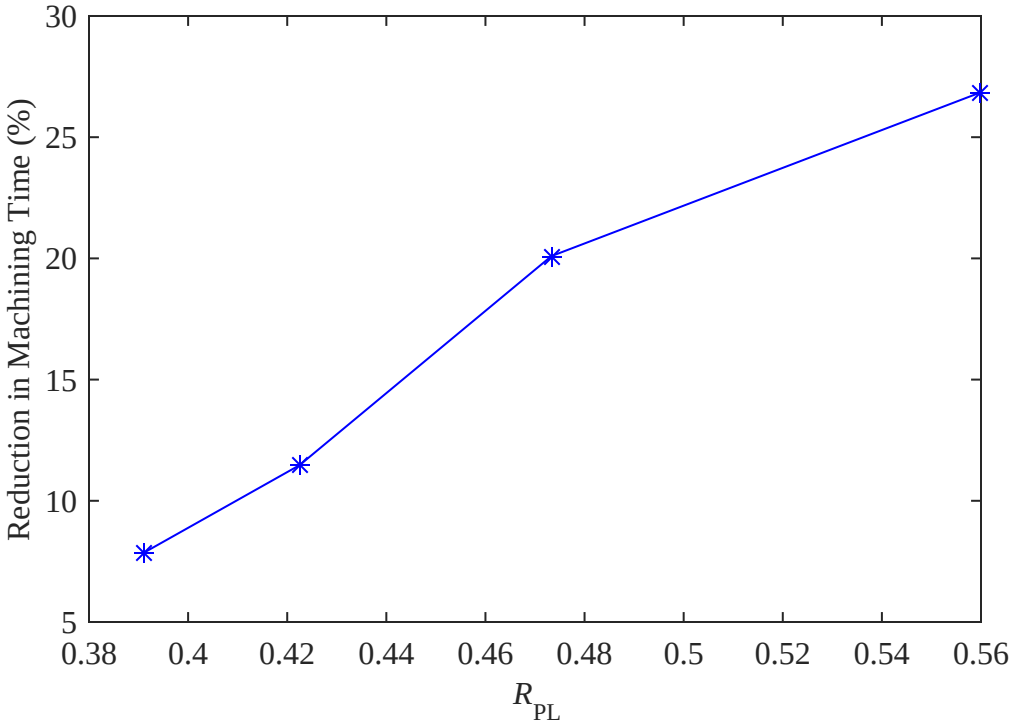


Figure 4.61: Directly-Controlled Machining Time Reduction as a Function of k_{CP}

tained by defining the path length ratio R_{PL} according to

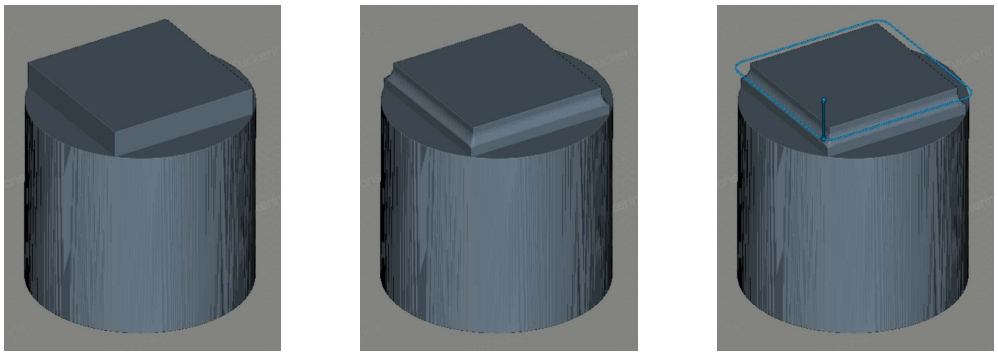
$$R_{PL} = \frac{L_{Cutting}}{L_{Total}} \tag{4.2}$$

where $L_{Cutting}$ is the length of the cutting only portion of the path (i.e., G1s) and L_{Total} is the total path length, including retractions (G0s). It is expected that higher values of R_{PL} will correspond to larger performance gains, as more of the path is dedicated to cutting in proportion to the total path length.

Figure 4.61 shows a plot of the percentage reduction in machining time as a function of R_{PL} for the four experimental toolpaths. This plot confirms that higher values of R_{PL} translate to larger reductions in machining time when using the direct servo control system, and enables a more meaningful comparison of the machining time improvements shown in Table 4.5.

4.6 Machining Results

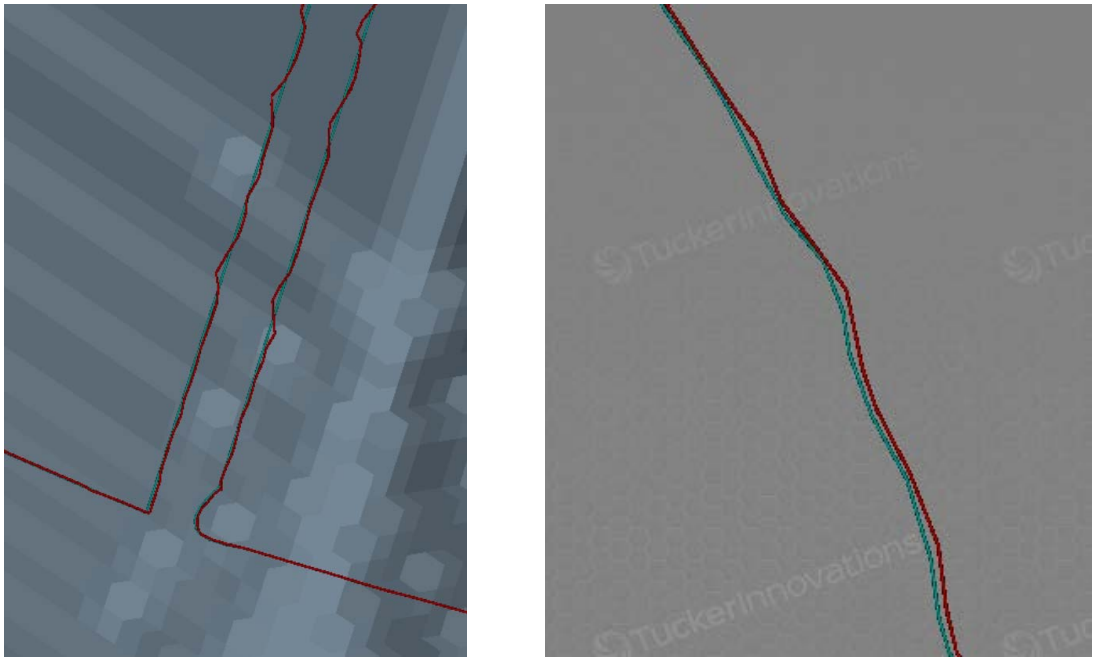
To evaluate the capability of the research machine tool to perform actual machining, a simple test part was designed and a series of test cuts were performed using the direct control system. The starting end ending volumes for the test part are shown in Figures 4.62(a) and 4.62(b), and the end volume with the toolpath overlaid is shown in Figure 4.62(c). The part was machined with a 0.25 inch diameter ball endmill.



(a) Start Volume (b) End Volume (c) Test Toolpath

Figure 4.62: Machining Test Part

The test toolpath was executed both with and without a cutting tool installed to enable comparison of axis positions when the cutting force is and is not exerted on the machine structure. Additional positional error and oscillation was encountered during actual machining as compared to air cutting, as shown in Figure 4.6. The actual toolpath position during the air cutting experiment is shown in teal, and the actual toolpath position during the machining experiment is shown in red. Figure 4.63(a) shows position oscillation about the retraction that was observed during cutting, and Figure 4.63(b) shows additional path error that was observed around a corner of the toolpath. The finished part, fixtured to the table of the research machine, is shown in Figure 4.64. A comparison of the axis position data from the air cut toolpath to the actual cut toolpath reveals an additional mean positioning error of 19.626μm along the entire length of the machined path.



(a) Tool Insertion

(b) Error Around Corner

Figure 4.63: Machining Test Part

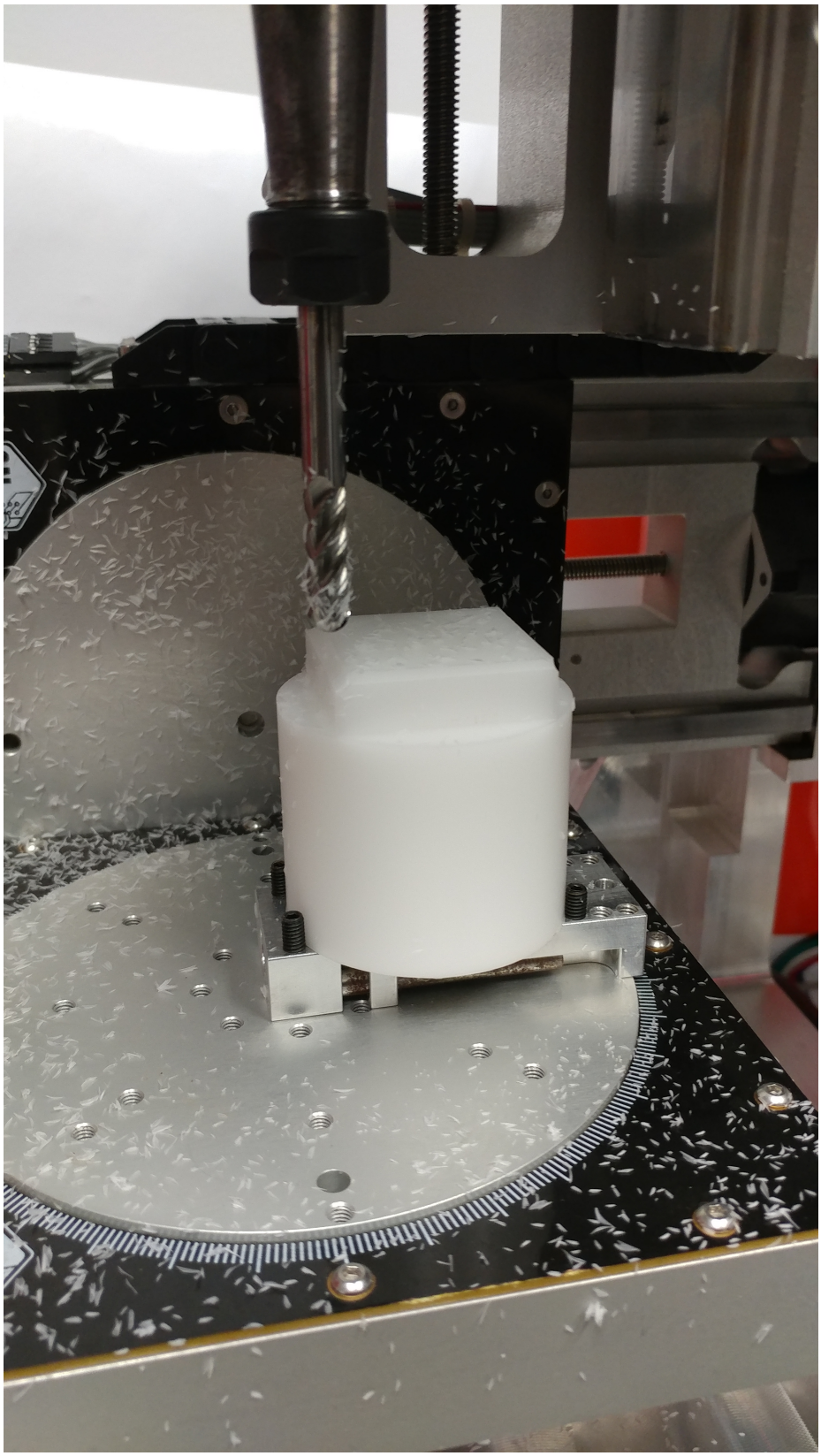


Figure 4.64: Test Part Fixtured to Table of Machine

4.7 Summary of Results

The results and analysis presented in this chapter demonstrate that the direct servo control system developed for this dissertation is capable of following time-optimal paths prescribed by the cloud trajectory planning system and generally achieves the target tool space positions, with some exceptions. Additionally, the performance of the interface application is sufficient for buffer level control, data collection, and interfacing with external devices. Finally, the direct servo control system demonstrates marked machining time improvements over the traditional G-Code programming approach.

However, there was some unpredictability observed in the experimental data, which can be attributed both to the lack of determinism in the interface application and to stepgen errors in Machinekit. For example, occasional drops of the trajectory buffer level to 0 that were encountered during execution of some of the paths is due to the fact that the interface application is not run on an RTOS, and therefore can unpredictably not be capable of sending servo positions with the appropriate frequency. However, this problem is relatively rare and could be remedied by both increasing the setpoint for the buffer fill level and increasing the maximum trajectory buffer size. Additionally, Machinekit sometimes commanded incorrect servo positions due to errors with step generation from the PRU. These incorrect positions were confirmed with encoder readings (as shown in Figure 4.60, where the actual path and the estimated path match) and can be attributed to incorrect step frequency from the PRU. Due to the structure of the servo loops in Machinekit, the control system was able to bring the axis position back to the commanded position shortly after the path deviation.

In summary, the experimental results indicate that the interlocking hardware and software components developed for this thesis work in concert to provide a working system that offers many benefits over traditional G-Code based machine tool controllers.

CHAPTER 5
DISCUSSION AND CONCLUSIONS

The goal of the research presented in this dissertation was to develop a general methodology for the control of motion profiles of a 5-axis CNC machine tool directly from a CAM system. The open-source machine tool hardware and accompanying software developed for this work are successful in accomplishing that goal, as demonstrated in Chapter 4.

5.1 Contributions and New Capabilities

The direct servo control architecture enables new capabilities that are not present in current CNC systems. These capabilities address the need for additional data flow between participants in the process planning and execution chain presented in Figure 1.2; specifically, the integrated CAM-CNC system developed and benchmarked in this work provides the following contributions:

5.1.1 Complete Control of Tool Trajectory

The fundamental difference between the direct control system and traditional G-Code based CNC systems is the transmission of servo loop setpoints instead of geometric primitives. In the system constructed for this dissertation, the servo loop setpoints completely define the tool trajectory, whereas geometric primitives do not convey the same amount of information. As a result, the trajectory of the cutting tool in the workpiece coordinate system can be completely controlled by the CAM system. This enables control of MRR throughout the toolpath when driving the cutting tool along an MRR-constrained trajectory.

5.1.2 A General Methodology for Collection and Storage of Dense Process Data

The interface application and communication pathways that were written to realize the direct control system enable process data collection from the machine tool at the servo rate of the motion control system, which was previously not practical with current implementations of machine tool networking standards. Additionally, the database written for the interface application is suitable for high frequency collection, storage, and analysis of this dense process data.

5.1.3 Cloud-Based Trajectory Planning

In contrast to typical CNC systems, where trajectory planning is performed online by the CNC system itself, the remote trajectory planning system used in this dissertation communicates with the interface application using a publicly-available websocket gateway. Rather than concentrating all computing power necessary for motion control of the machine tool into a local system, the cloud-based trajectory planner provides an architecture where remotely-managed HPC capability can be utilized for motion control tasks. As a result, multiple directly-controlled machine tools could rely on the same trajectory planning system, which enables easier manageability and upgradability.

5.1.4 Dense Feedback to the Process Planning System

The interface application developed for this dissertation serves as the critical link between the CNC machine tool and the CAM system that is not present in traditional machine tool controller deployments and which is conspicuously absent from the process planning and execution chain shown in Figure 1.2. The interface application provides a means by which a CAM system can connect to the data output of the machine tool, which enables near-realtime visualization and analysis of the realized tool trajectory and accompanying auxiliary data. Figure 5.1 demonstrates this capability, specifically developed for the SculptPrint CAM system: the voxel point samples are shown in light blue, the planned trajectory is

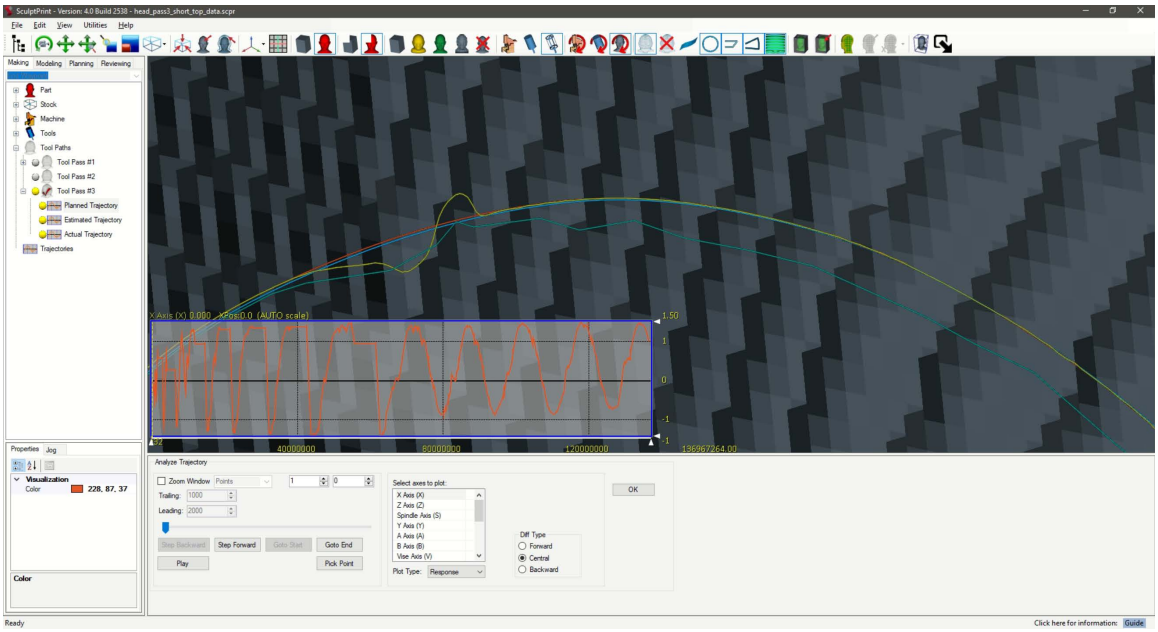


Figure 5.1: Toolpath Defect Analysis in SculptPrint

shown in burnt orange, the estimated trajectory is shown in yellow, and the actually realized trajectory is shown in teal. The voxels of the part model are visible as grey cubes. As shown in the Figure, data collected during execution of a toolpath by the machine tool system developed for this work enables visualization of toolpath errors that would previously have gone unrecorded by a typical, closed-architecture machine tool system.

5.1.5 Enhanced Toolpath Traversal Speed

Through time-optimal trajectory planning and the use of an offline TP, the direct servo control strategy was able to demonstrate vast reductions in toolpath traversal time for each of the experimental cases as compared with standard G-Code toolpaths. This is an extremely important development for manufacturers, where even minor improvements in the speed of a machining operation can directly result in substantial cost savings.

5.1.6 Enhanced Usability for Complex Machine Tool Systems

The use of the direct control approach enables execution of complex machining process plans without the need for experience in G-Code programming, which can be a major limitation to users of CNC machine tools. In contrast to typical machine tool systems, which require post-processing and occasional debugging of G-Code to create a working part program, the direct control system allows a user to run a 5-axis toolpath directly from the CAM system in a way similar to the operation of a 3D printer.

5.1.7 A Fully Instrumented and Open-Source Research Desktop Machine Tool

The machine tool hardware developed for this research is based on fully open-source platforms that provide a framework for machine tool research that would not be possible with traditional machine tools. Due to the proprietary nature of currently available machine tool systems, instrumentation and high-frequency data collection are difficult; however, the system developed for this work is fully accessible and enables a wide range of research activities.

5.2 Answers to Research Questions

The research questions posed in Section 1.8.2 are addressed below using knowledge gained from this research.

1. Control of RT components from the non-RT interface application and CAM system requires extensive use of buffering to absorb non-deterministic latency between the systems. The proportional controller used to control the buffer level is necessary to ensure that the frequency of servo loop setpoint transmission is adjusted to maintain a target level in the trajectory buffer. The proportional controller generally is capable of maintaining the buffer above empty, though occasional dips were observed during experimentation that could cause a cessation of motion. In general, the RT control

system can be commanded effectively from the non-RT subsystems presented in this dissertation if the user is willing to accept some delay between command issue and command execution (e.g., the delay between when a position setpoint command was issued and when the axis actually reached the setpoint, as shown in the joint space data plots).

2. This dissertation demonstrated the use of a cloud-based trajectory planning system that is capable of easing the computational load on the interface application. The planning system received position samples and created motion trajectories using those samples, and then relayed the completed trajectories back to the interface application through a websocket. The success in this implementation demonstrates that a cloud-based computing system can successfully be employed for critical elements in a machine tool control application.

3. Position sampling by the CAM system does not have an effect on the programmed tool velocity, as it would with traditional G-Code programming. The density of position sampling simply affects the number of samples that are transmitted to the TP; the TP in turn returns trajectories that are sampled temporally at the servo rate of the machine tool. As a result, spacing between temporal samples is actually the factor that determines the velocity of the cutting tool along the toolpath, which is completely independent of the spatial sampling rate that is presented to the TP.

4. As demonstrated in Section 4.5, offline planning of trajectories using the cloud-based TP enables substantial machining time reduction that is positively correlated with the path length ratio. These gains are due both to an improved trajectory planning strategy and a control architecture that removes the need for a finite lookahead buffer. Elimination of the velocity saturation limitation will enable execution of MRR-controlled toolpaths that were previously not feasible with the traditional approach.

5. Process data feedback to the CAM system enables rich visualization and analysis of positional derivatives of the axis actuators, and also allows for inspection of toolpath errors in WPC. This information can be leveraged by toolpath designers and manufacturing engineers to modify the toolpath itself or tune the kinematic limits presented to the TP. Upon repeated executions of the tuned toolpaths, personnel can optimize toolpath trajectories to achieve certain benchmarks, such as execution time or positional accuracy.

6. As demonstrated in the data analysis, measured tool space deviation from the planned trajectory is, on average, approximately one voxel (when using 50μm voxel size, which is standard practice in SculptPrint). It is important to note that this value is determined by the axis actuator position, which does not account for physical errors of the machine tool. At times, the positional deviation is multiple voxels (e.g., in the case of the head top toolpath, the maximum positional deviation was approximately five voxels in size). However, given the average positional deviation, it is safe to say that generally the position of the tool tip can be controlled to within a single voxel for each experimental toolpath.

5.3 Future Work

The hardware and software developed for this dissertation provide an excellent platform for further research into machine tool control, data analysis, and functionality. The series of upgrades to the machine tool presented here should be performed to study their effects on system performance. Specifically,

1. *Online Trajectory Optimization*: This dissertation performed static offline trajectory planning, where the generated trajectories depend only on the position samples from the CAM system and the kinematic limits of the machine tool. The machine control software and accompanying trajectory planning system should be modified to be

capable of replanning trajectories in response to machining conditions, such as feed rate override or cutting power data collected during machining. This capability would enable iterative trajectory optimization based on experience garnered from execution of a toolpath.

2. *Replacement of Axis Stepper Motors with 3-Phase AC Servomotors*: The stepper motors used as actuators for each axis suffer from low maximum velocity and noisy operation. In a future version of this machine tool, those actuators should be 3-phase AC servomotors to provide both higher performance and more precise control. Additionally, the use of servomotors may alleviate some of the noise that was observed on the encoder signals, as torque production of 3-phase motors is smoother than that of stepper motors.

3. *Spindle Axis Position Control*: Development of position control capabilities for additional axes, such as the spindle axis, will enable exploration of rigid tapping and other coordinated motion that requires positioning of the spindle. Additionally, cutting power can be studied at different rotational positions of the endmill, which will enable more precise realtime simulation of stress on the cutting tool as it rotates.

4. *Implementation of Hybrid Manufacturing Capability*: The spindle on the machine tool developed for this research is hollow and was designed to allow polymer filament to pass through it. This polymer filament can be used for a fused deposition modeling (FDM) process, which would turn the machine tool into a hybrid manufacturing platform. Implementation of functional hybrid manufacturing capability would also require the addition of another axis, the extruder axis, that controls the feed of the filament. The use of the direct control system on a hybrid manufacturing platform would enable 3D printer-like operation for both additive and subtractive manufacturing processes on the same machine.

5. *MRR-Based Trajectory Control*: The goal of this research was to develop the frame-

work necessary to control cutting tool motion along arbitrary trajectories that are determined by a remote trajectory planning system instead of by G-Code. For this work, the trajectory planner used standard time-optimal path planning whose constraints consisted of the axis kinematic limits of the research machine tool. A time optimal trajectory planning strategy that incorporates WPC velocity constraints based on the MRR limit of the cutting tool in use should be developed to enable cutting tool velocity along the workpiece surface to be constrained to not exceed the MRR limit.

6. *Use of Commercial Cloud Services*: Elements of the interface application, including the trajectory planning system, can be deployed on a commercial cloud services platform, such as Amazon Web Services (AWS), which will provide a means to further explore remote HPC capability in both trajectory planning and realtime simulation of the machining process. One primary interest area is to determine how much processing capability should be kept at the level local to the machine tool, and how much can be offloaded to a remote platform.

7. *Integration of Additional Material Handling Equipment*: The emergence of low-cost open source robotic platforms opens up opportunities for integration of automatic part loading and unloading systems. The robotic platform could be integrated through the interface application, which would enable data flow between the robot and the machine tool.

8. *Deployment of Additional Sensors*: In the machine tool presented in this work, the tool tip position in the workpiece coordinate system is determined from joint position sensor (encoder) measurements that are transformed using the FKT. Another avenue of exploration is the use of sensors on the tool, the table, or workpiece itself, such as inertial measurement units (IMUs), that offer additional measuements that can be fused with joint space measurements. Sensors on the tool, table, or workpiece would be immune from issues with backlash or deformation of axis ballscrews, and could

provide important insight into positioning errors of the machine tool.

9. *Implementation on an Industrial Machine Tool*: Scaling up the direct control system to a larger machine tool is necessary to further demonstrate CAM-CNC integration on industrial-scale equipment. Work is already underway in retrofitting a 30-year old machine tool with modern servomotors and drives, and the direct control system will be deployed on this machine tool once mechanical and electrical upgrades are complete. The machine to be used for this work is shown in Figure 5.2.

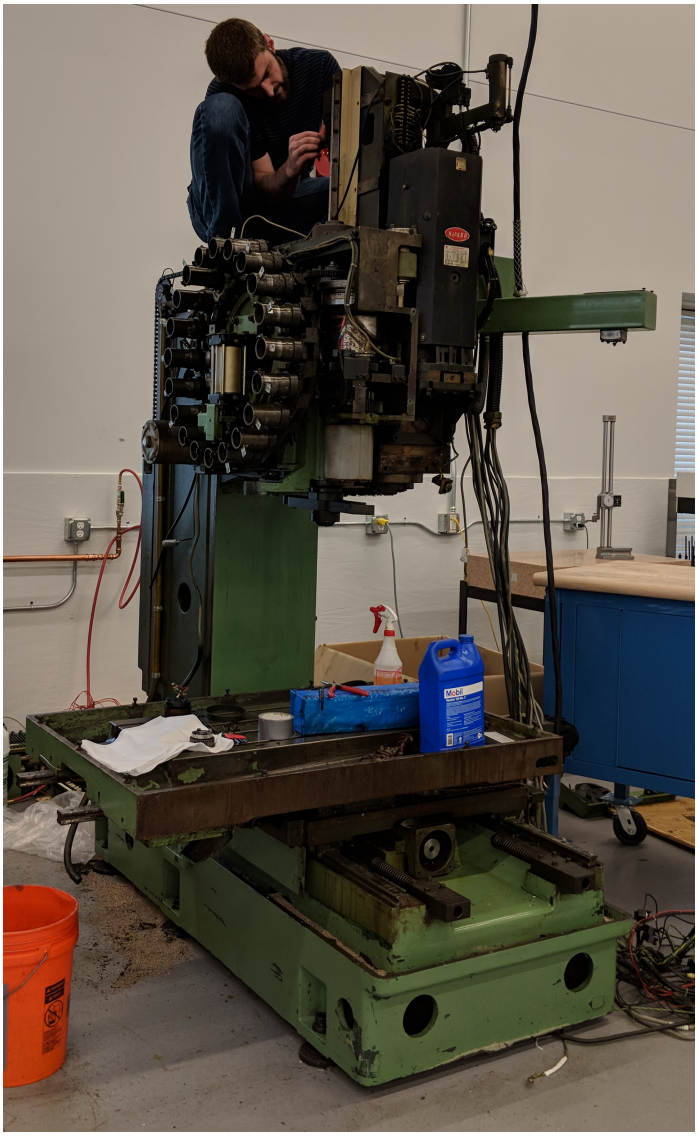


Figure 5.2: Retrofit of Industrial Machine Tool

Appendices

APPENDIX A

JOINT SPACE DATA FOR EXPERIMENTAL TOOLPATHS

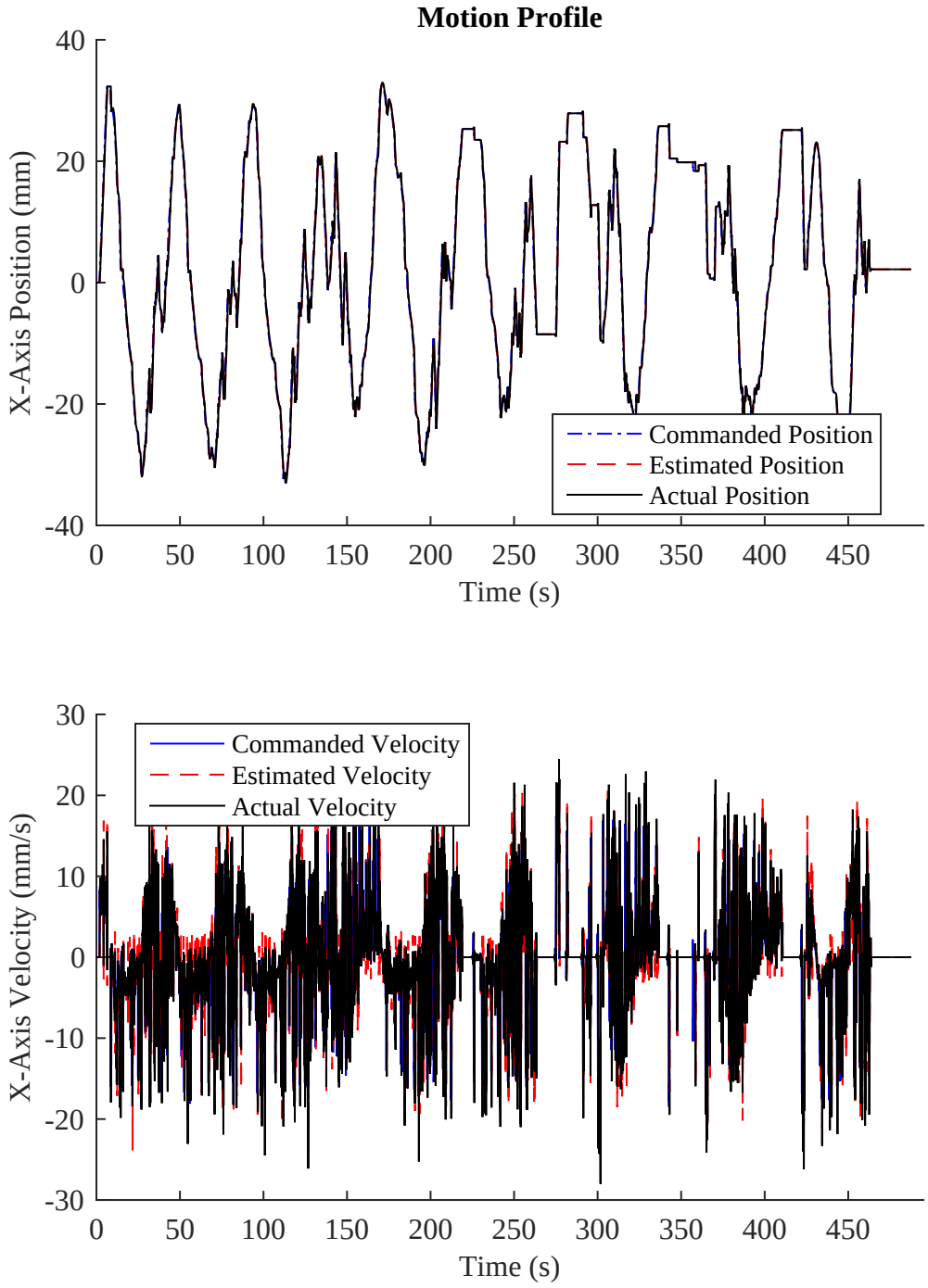


Figure A.1: X-axis Position and Velocity Progression for Head Bottom Toolpath

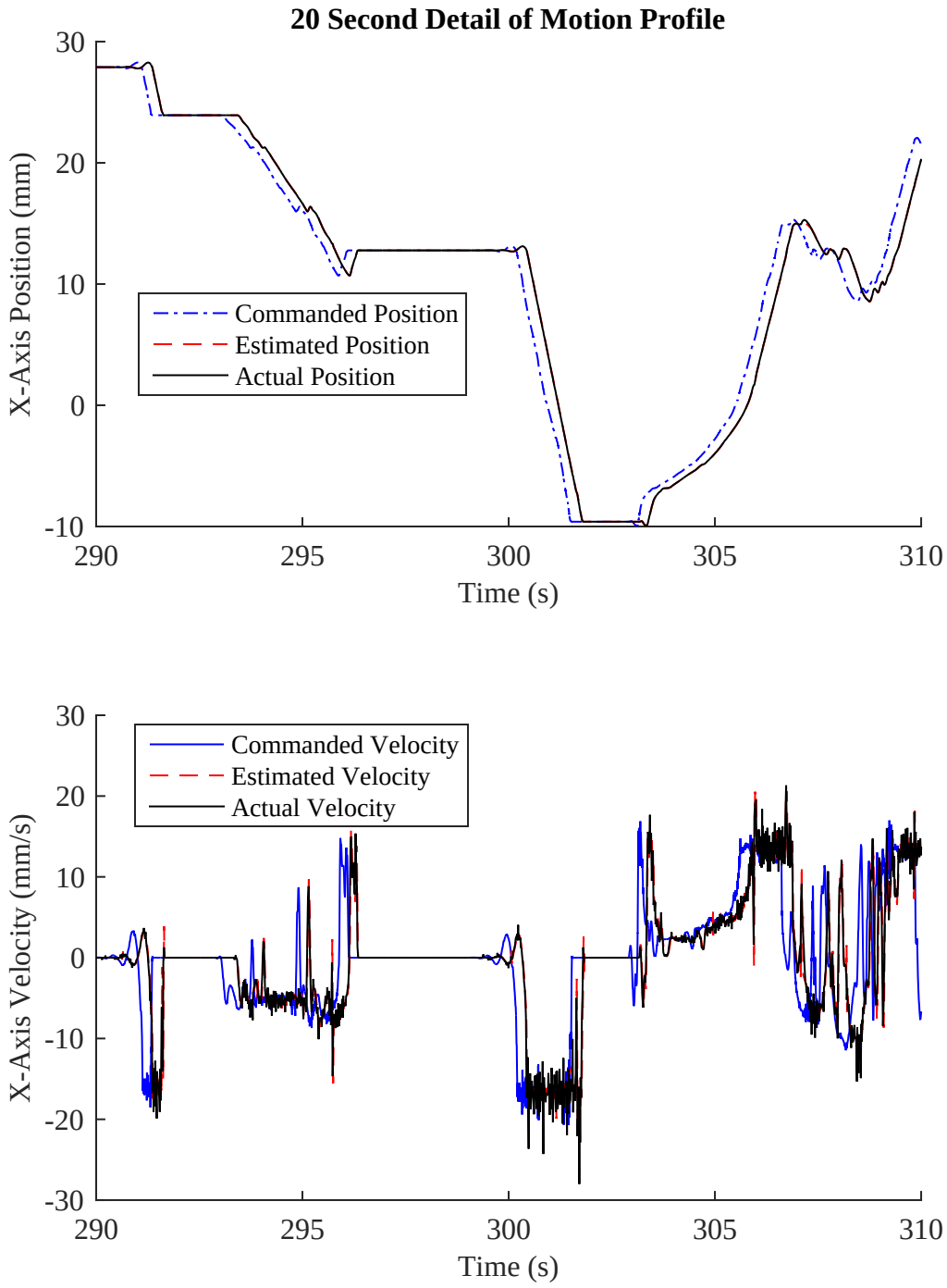


Figure A.2: 20 Second X-axis Position and Velocity Progression for Head Bottom Toolpath

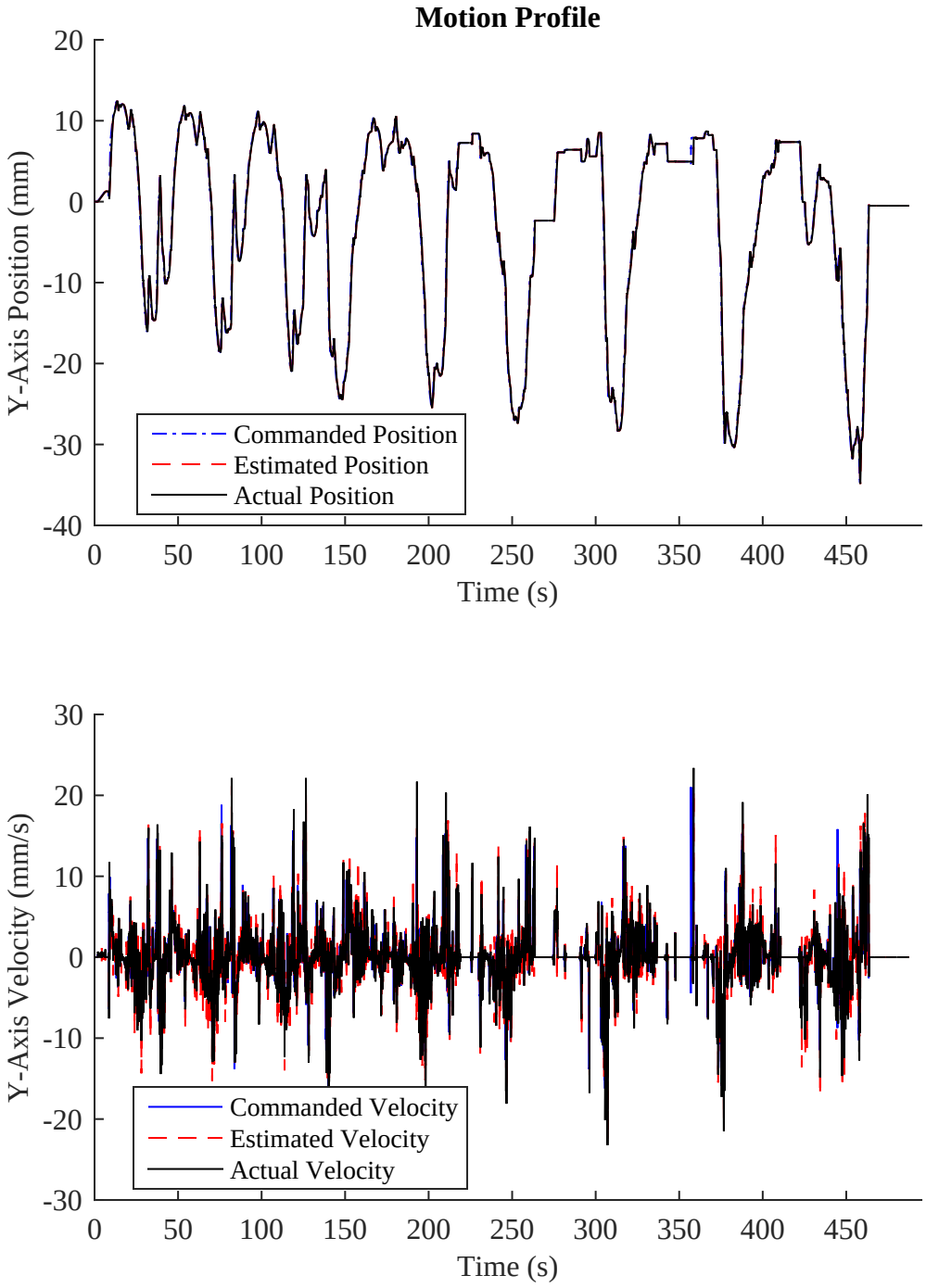


Figure A.3: Y-axis Position and Velocity Progression for Head Bottom Toolpath

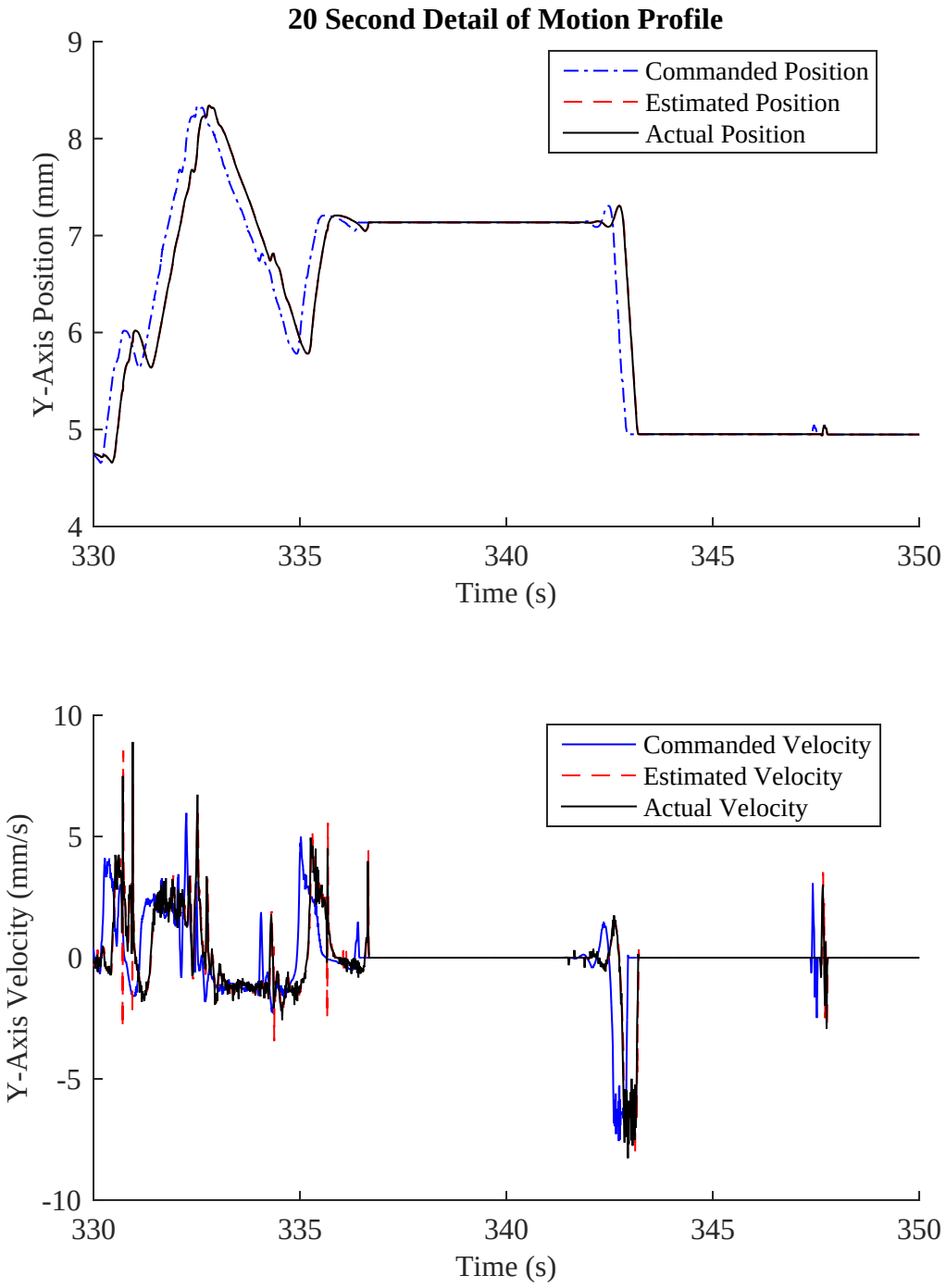


Figure A.4: 20 Second Y-axis Position and Velocity Progression for Head Bottom Toolpath

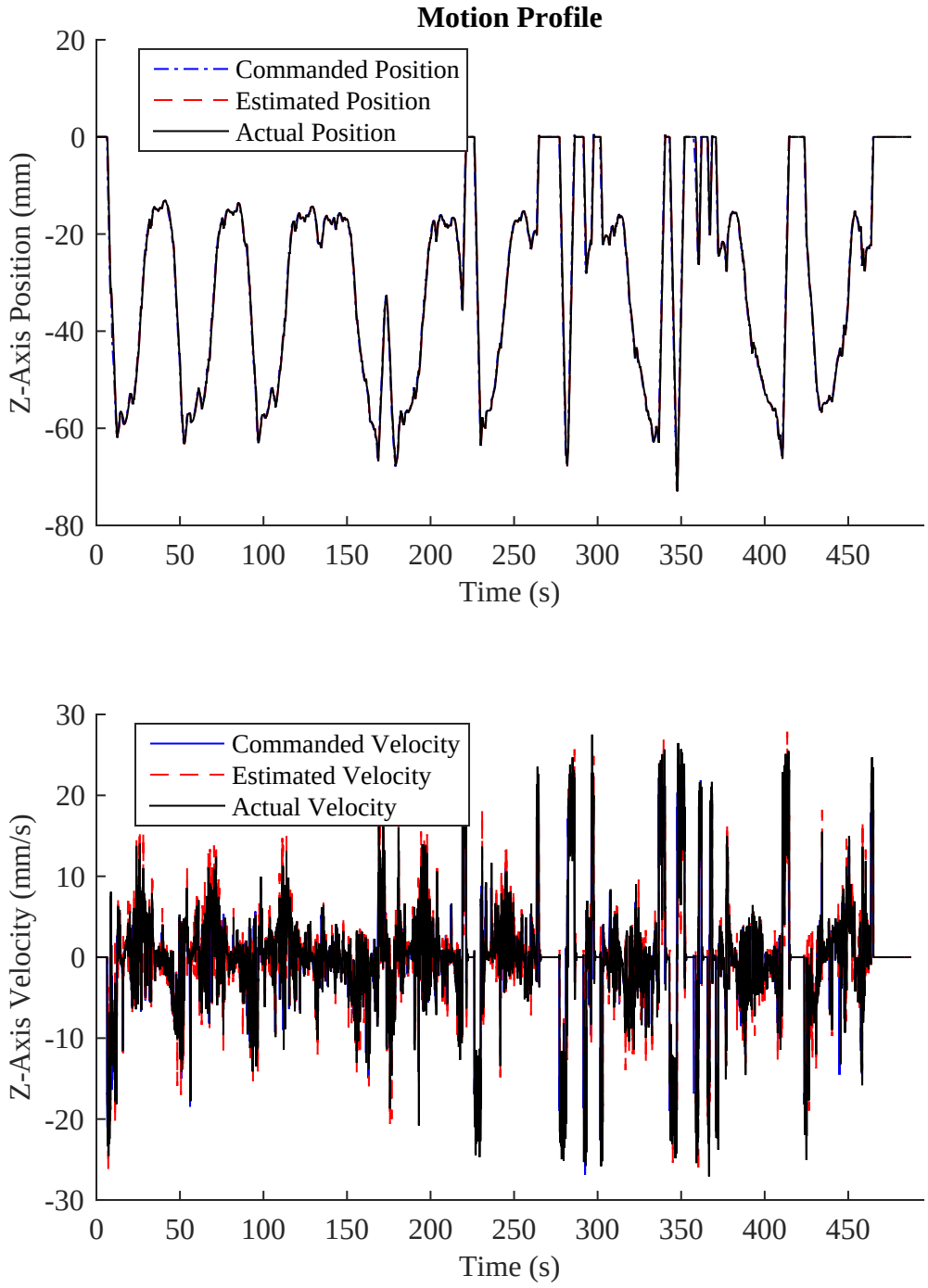


Figure A.5: Z-axis Position and Velocity Progression for Head Top Toolpath

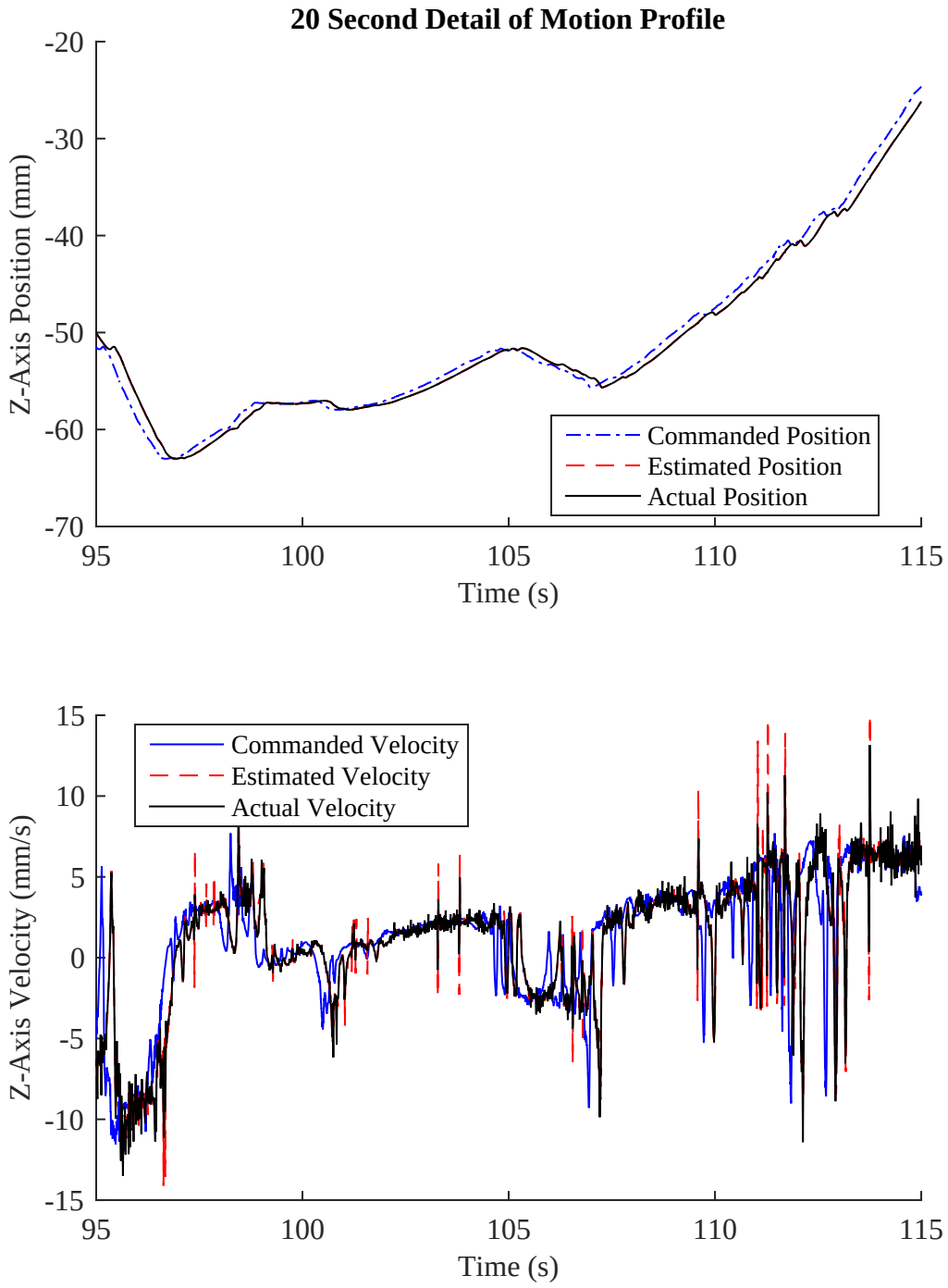


Figure A.6: 20 Second Z-axis Position and Velocity Progression for Head Bottom Toolpath

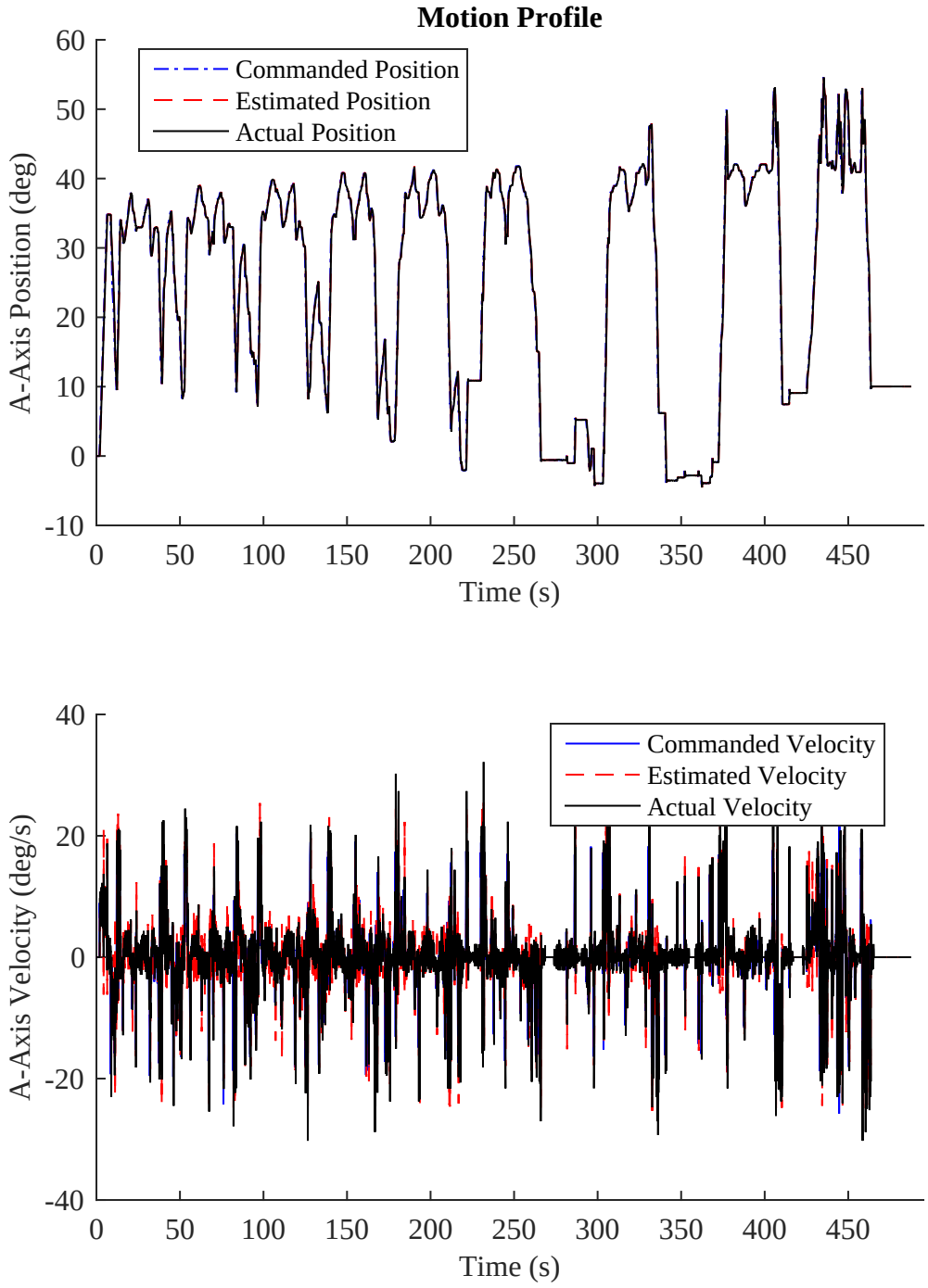


Figure A.7: A-axis Position and Velocity Progression for Head Bottom Toolpath

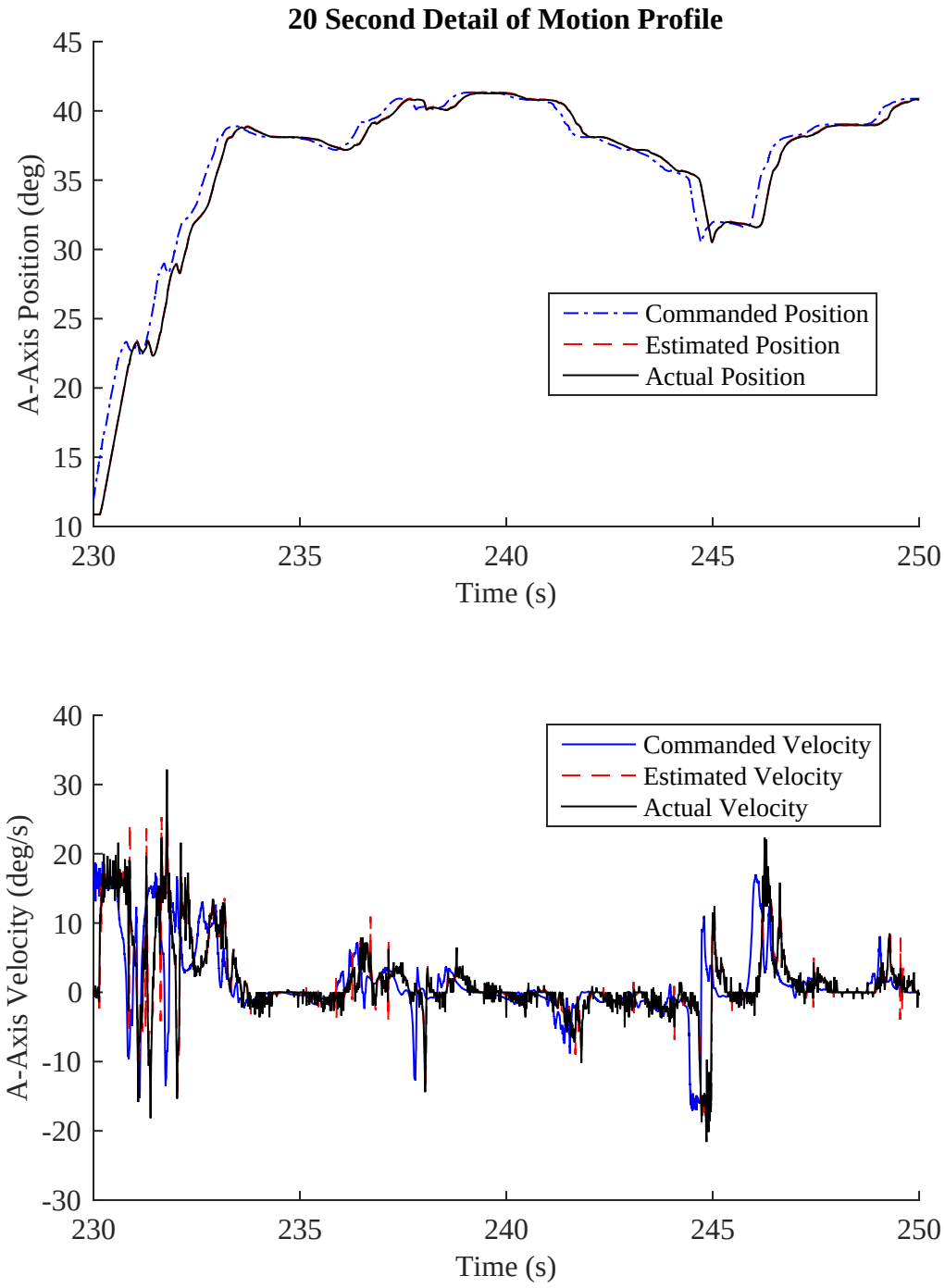


Figure A.8: 20 Second A-axis Position and Velocity Progression for Head Bottom Toolpath

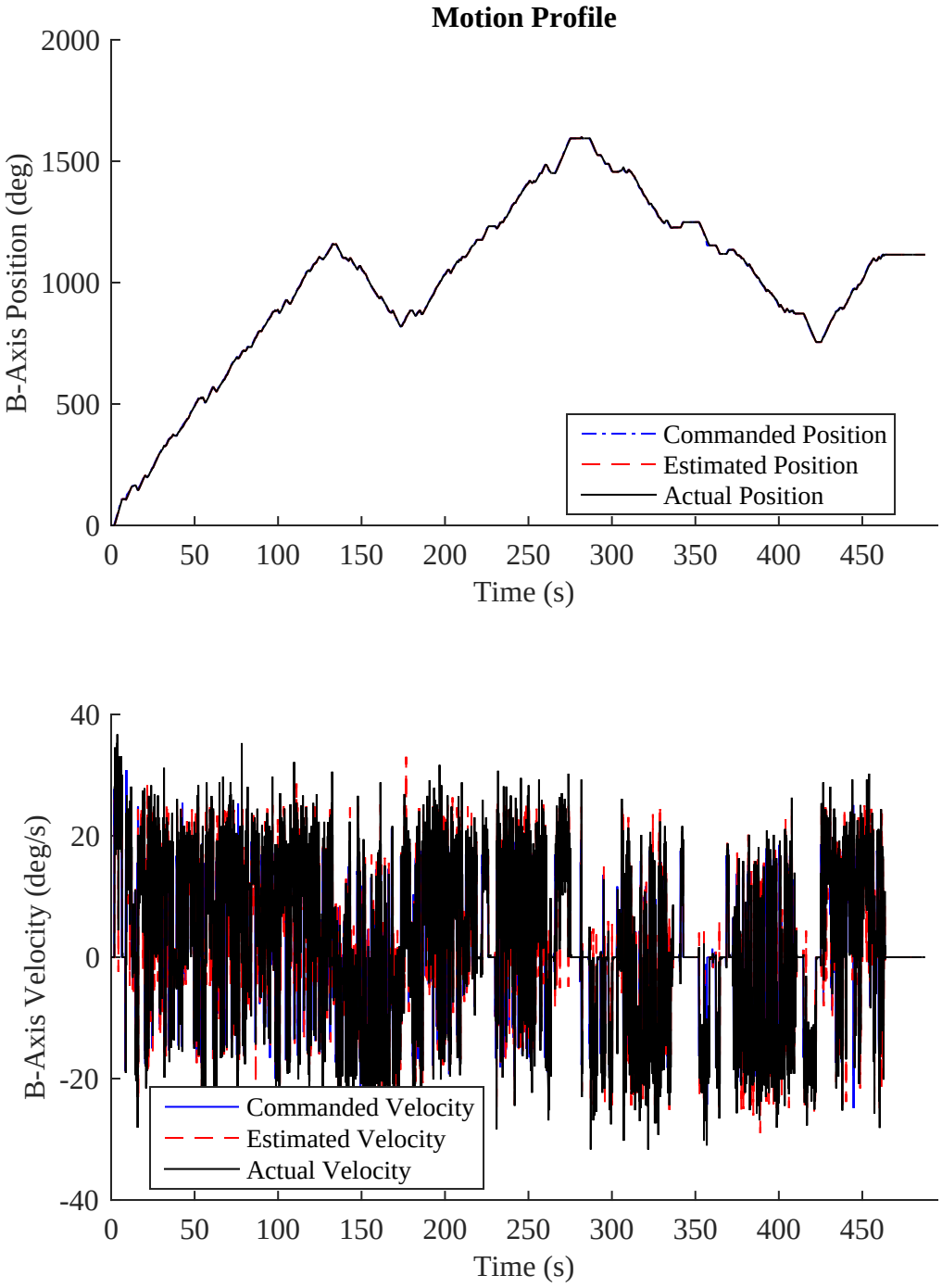


Figure A.9: B-axis Position and Velocity Progression for Head Bottom Toolpath

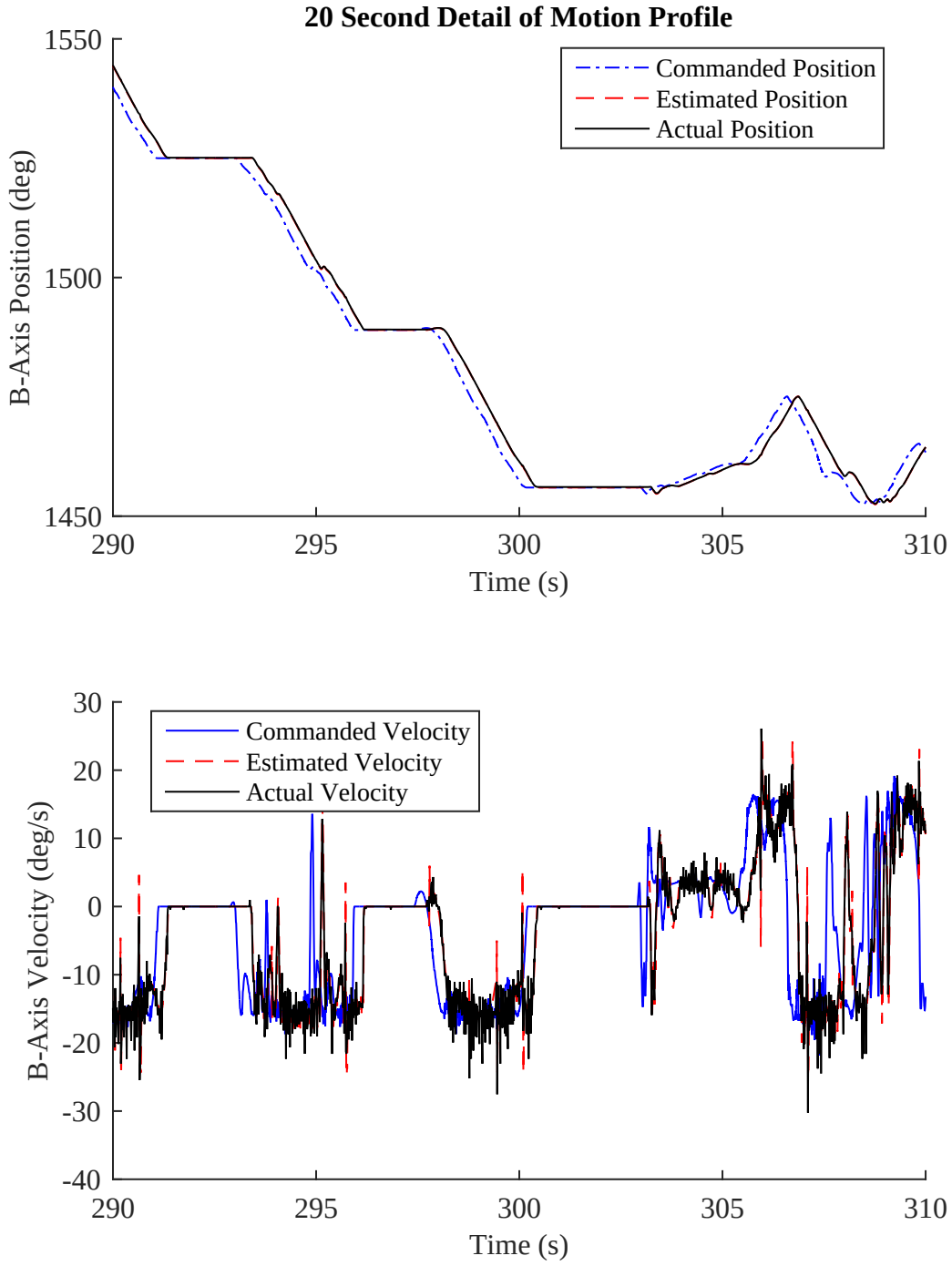


Figure A.10: 20 Second B-axis Position and Velocity Progression for Head Bottom Toolpath

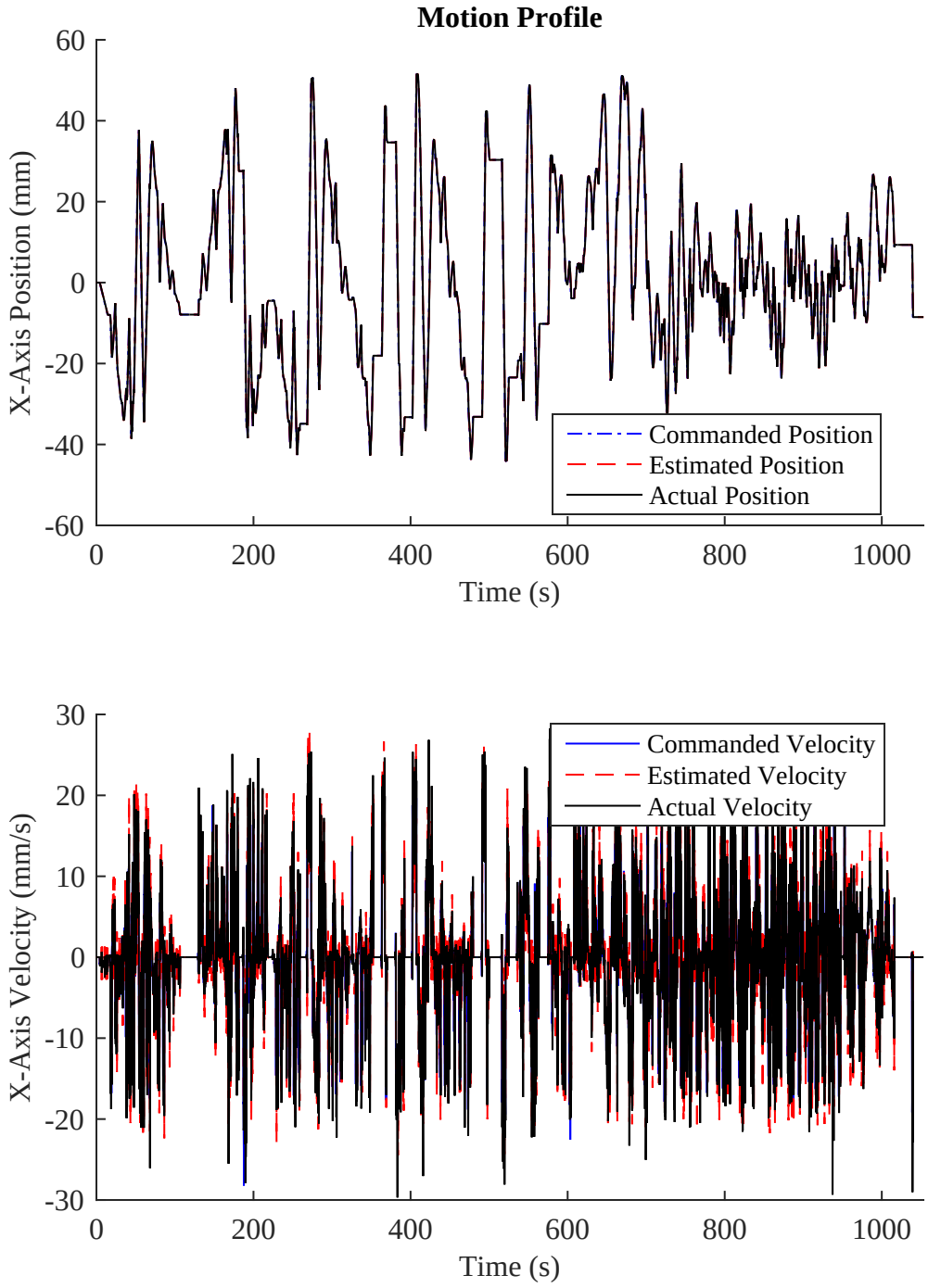


Figure A.11: X-axis Position and Velocity Progression for Candleholder Top Toolpath

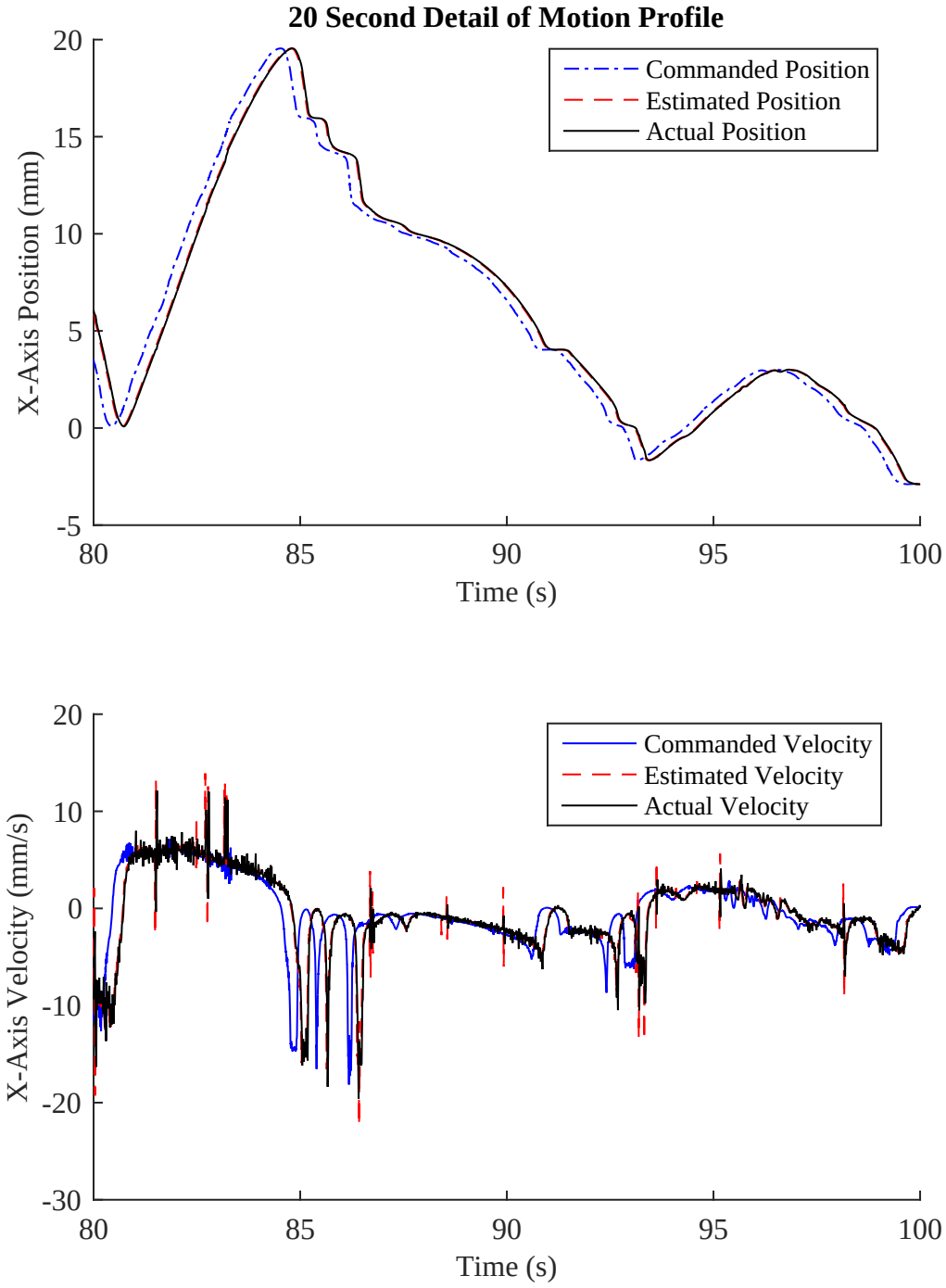


Figure A.12: 20 Second X-axis Position and Velocity Progression for Candleholder Top Toolpath

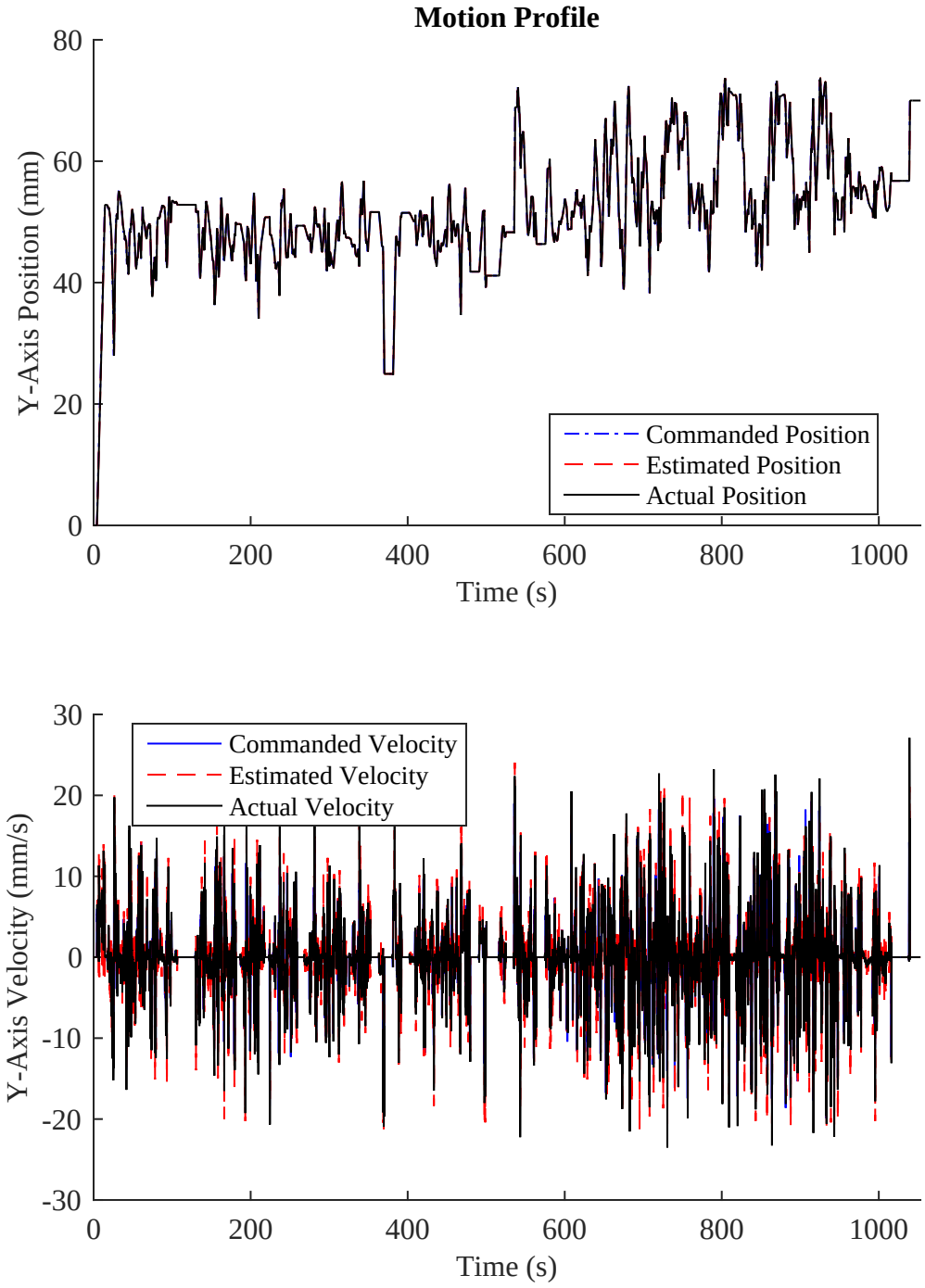


Figure A.13: Y-axis Position and Velocity Progression for Candleholder Top Toolpath

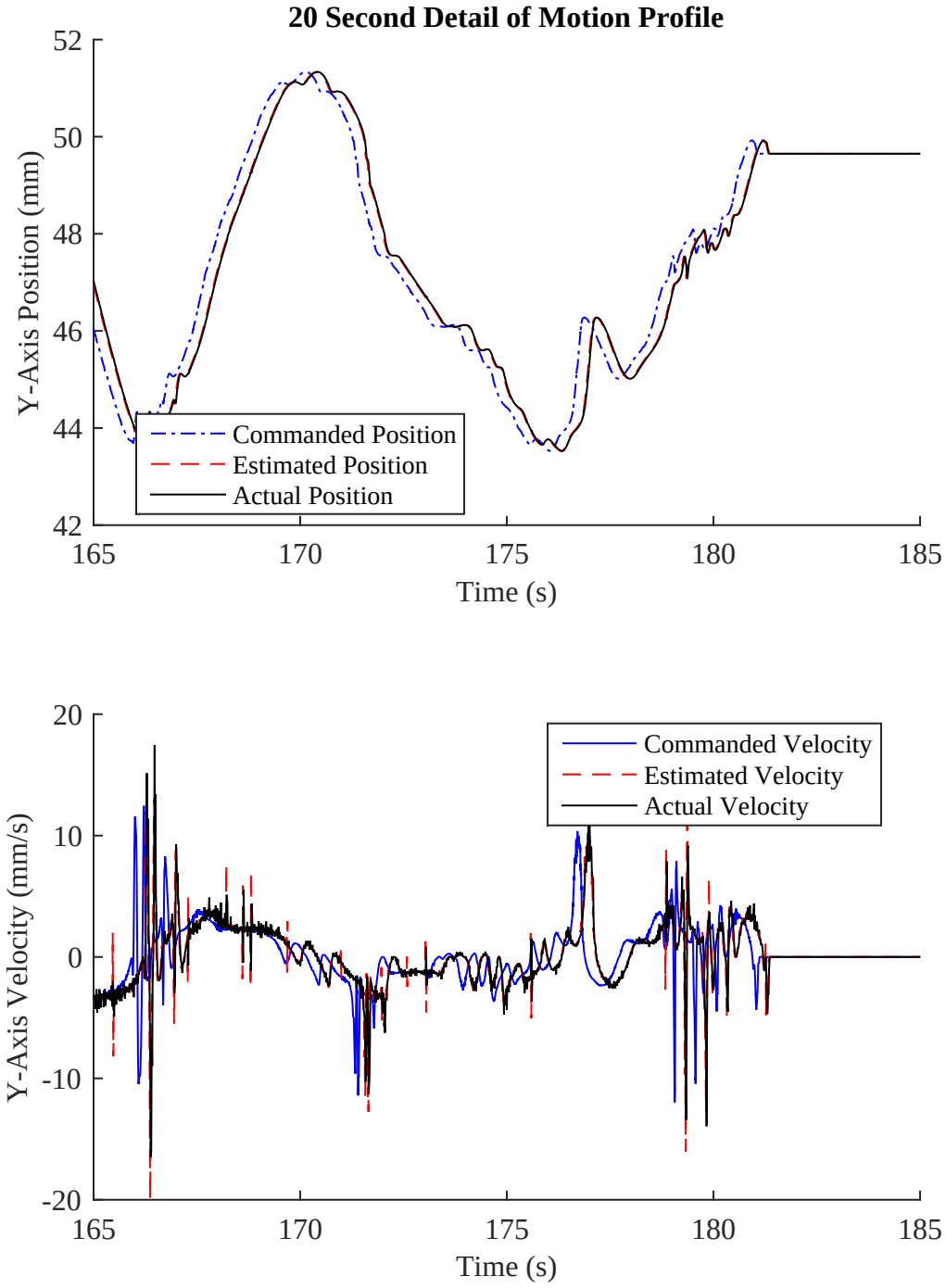


Figure A.14: 20 Second Y-axis Position and Velocity Progression for Candleholder Top Toolpath

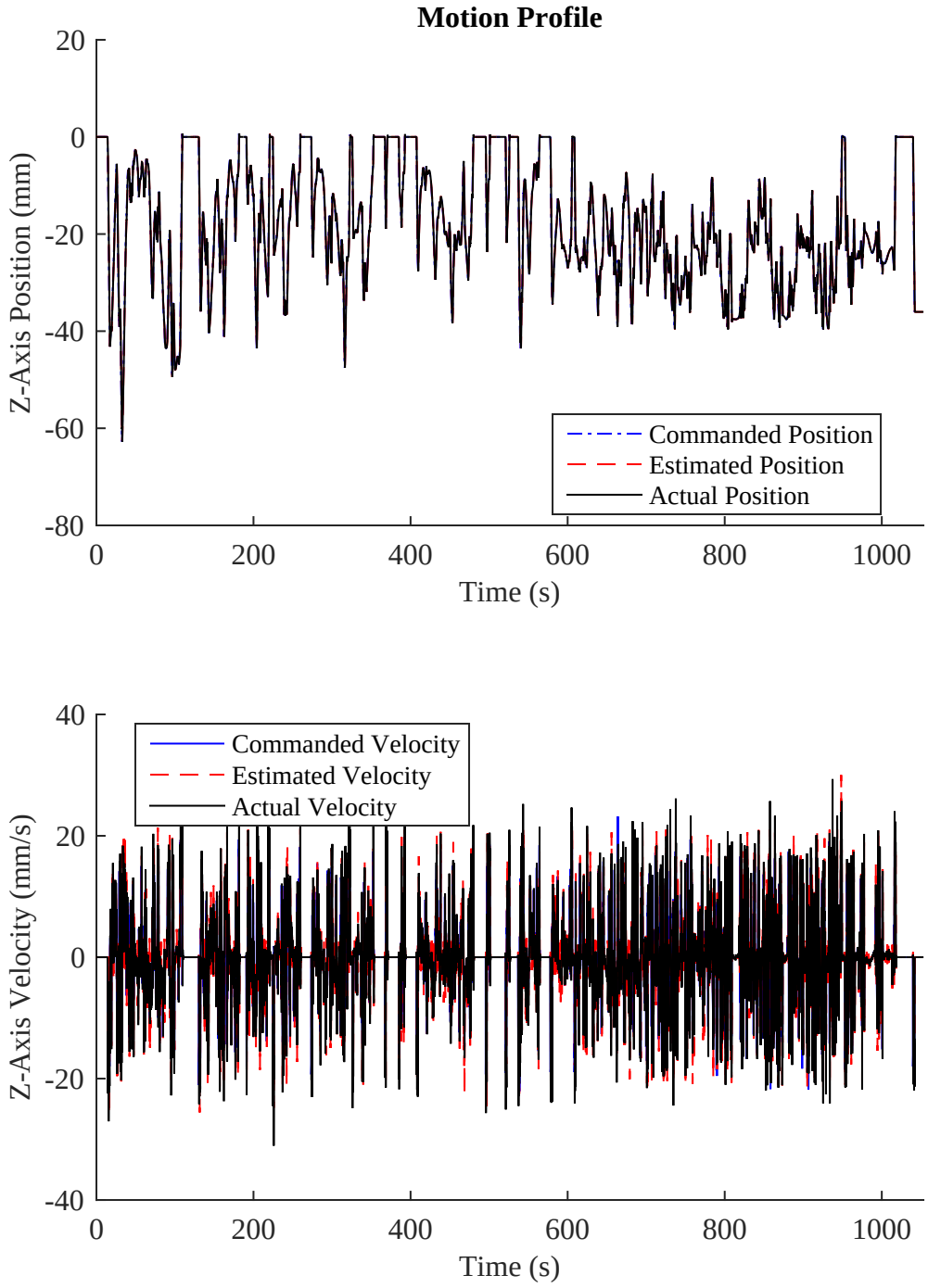


Figure A.15: Z-axis Position and Velocity Progression for Candleholder Top Toolpath

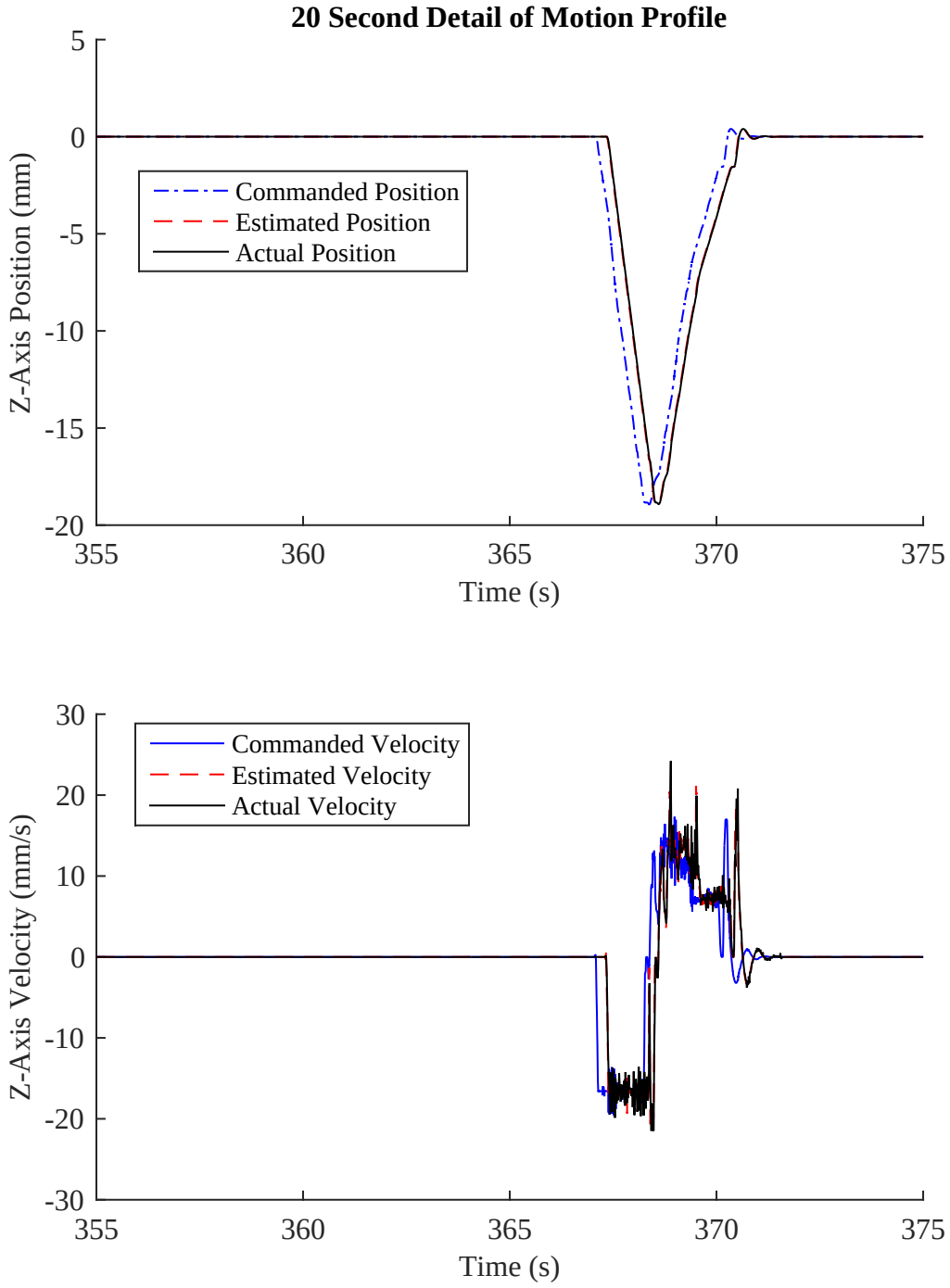


Figure A.16: 20 Second Z-axis Position and Velocity Progression for Candleholder Top Toolpath

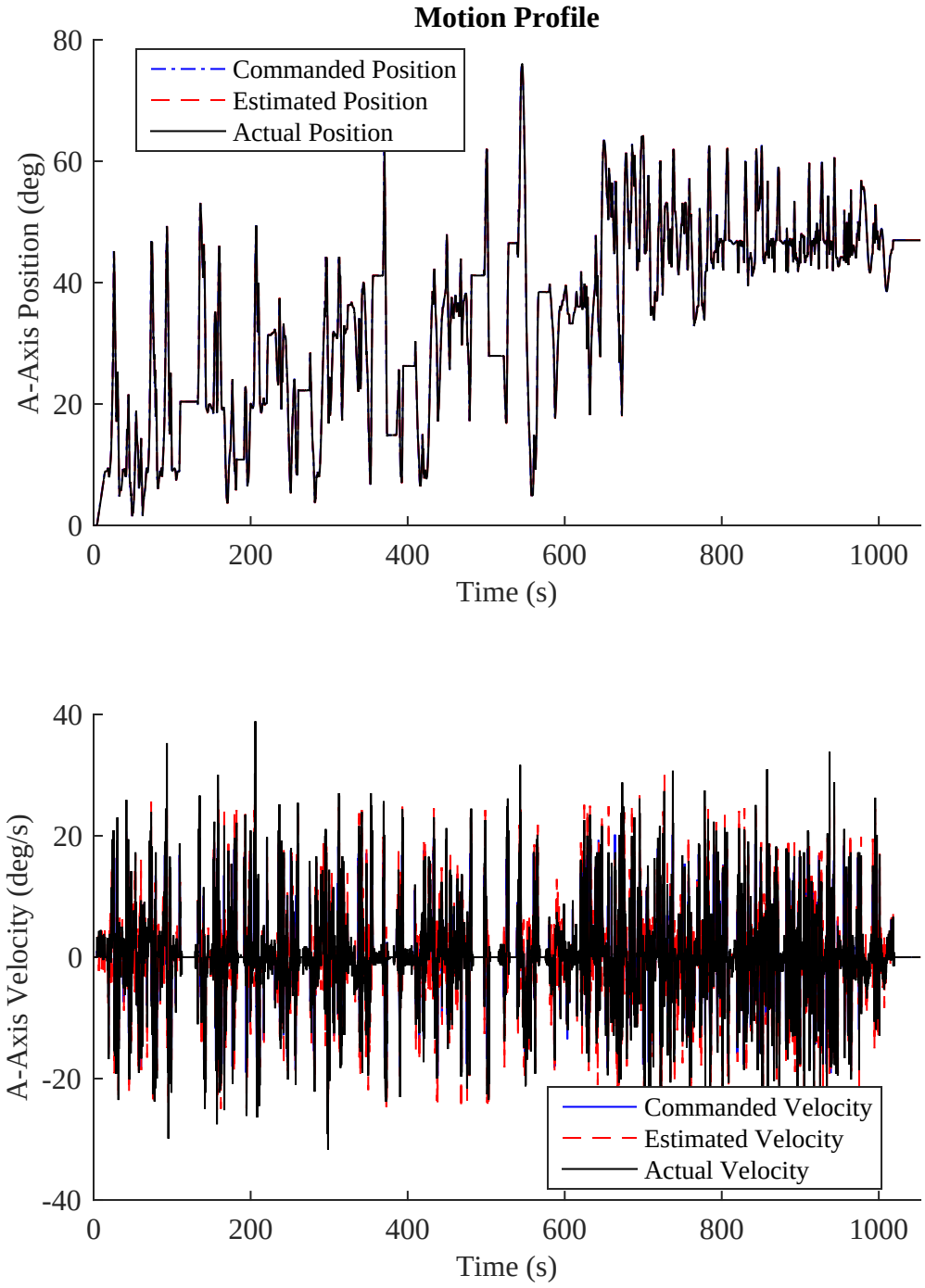


Figure A.17: A-axis Position and Velocity Progression for Candleholder Top Toolpath

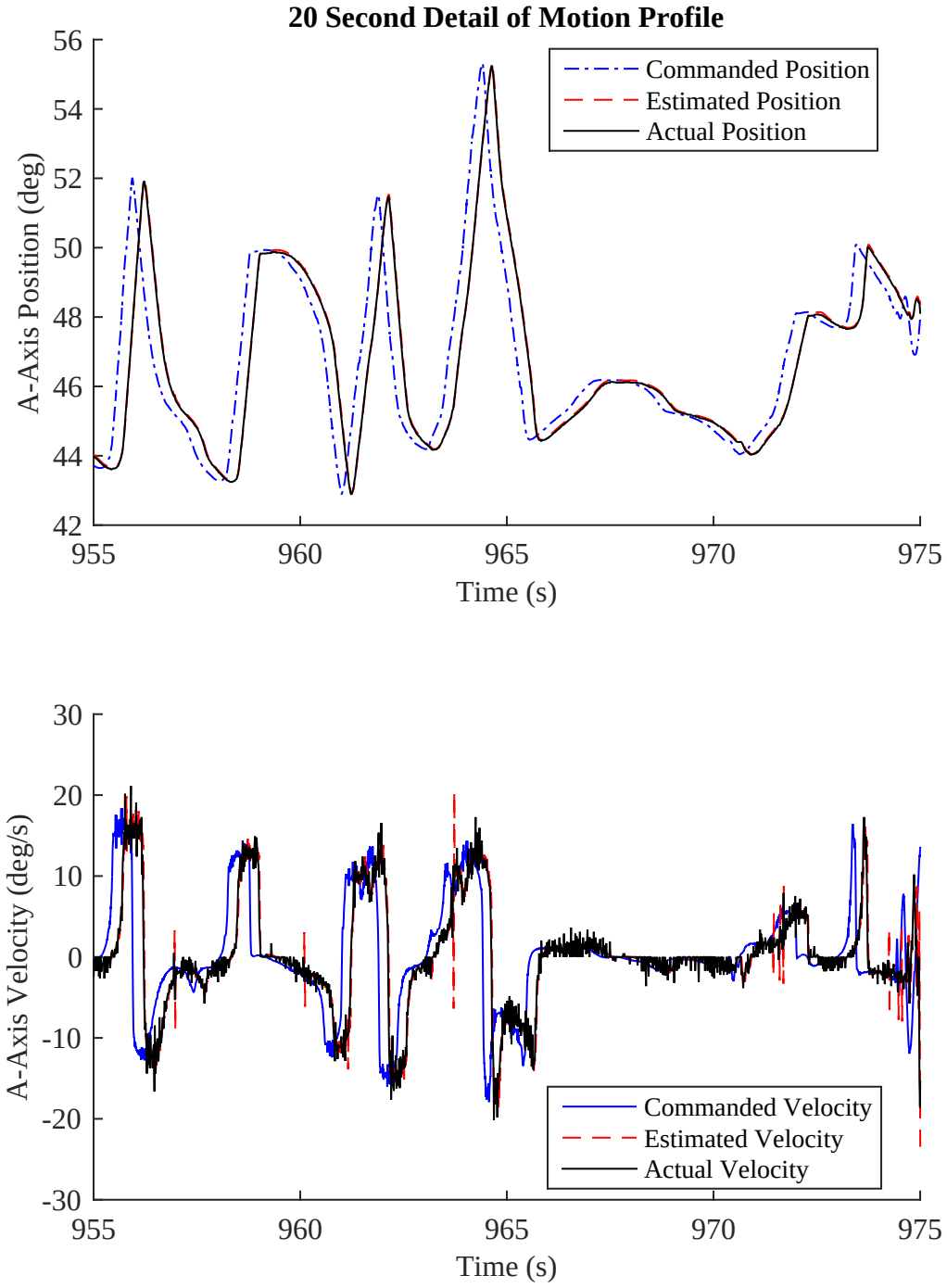


Figure A.18: 20 Second A-axis Position and Velocity Progression for Candleholder Top Toolpath

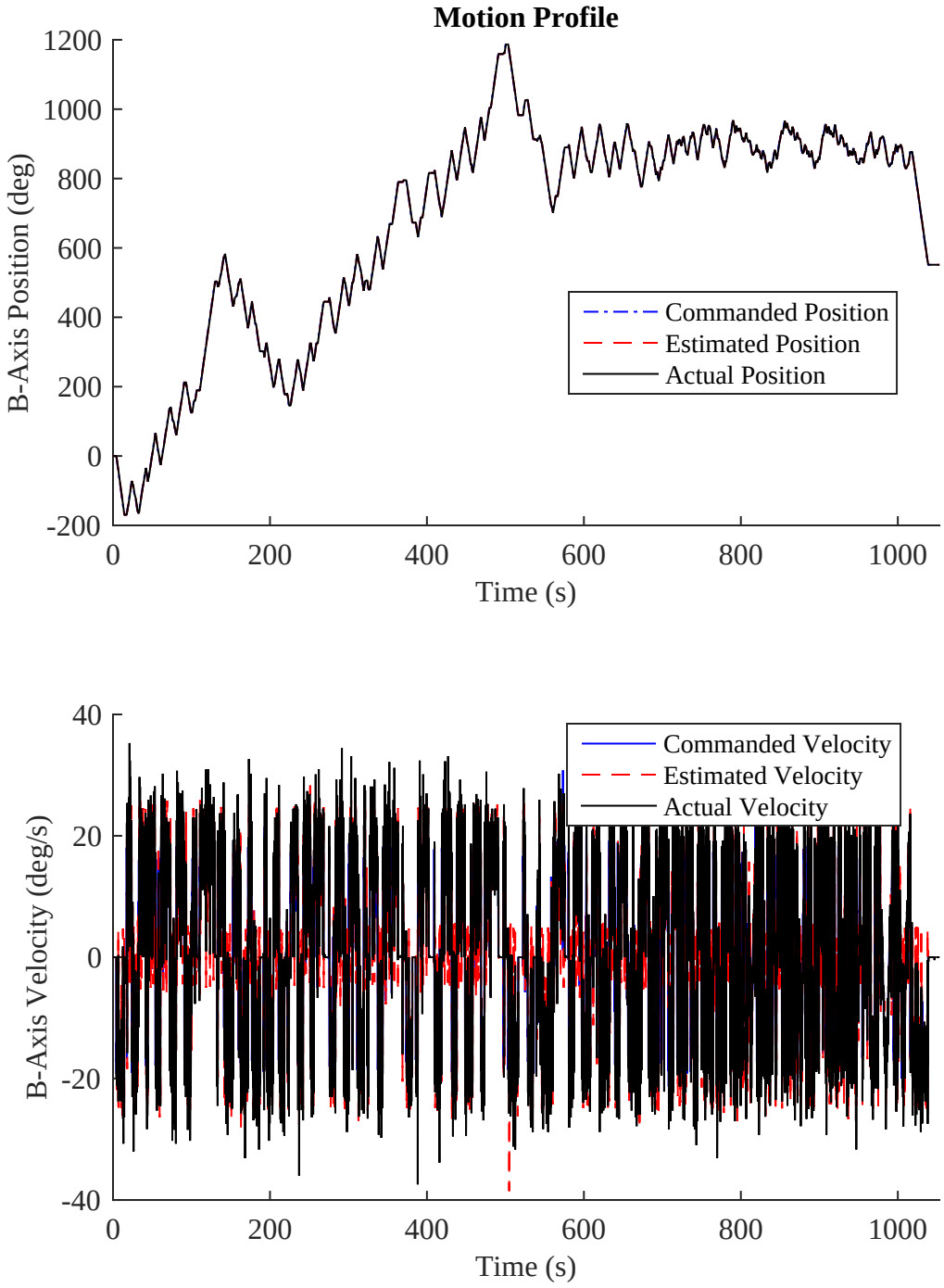


Figure A.19: B-axis Position and Velocity Progression for Candleholder Top Toolpath

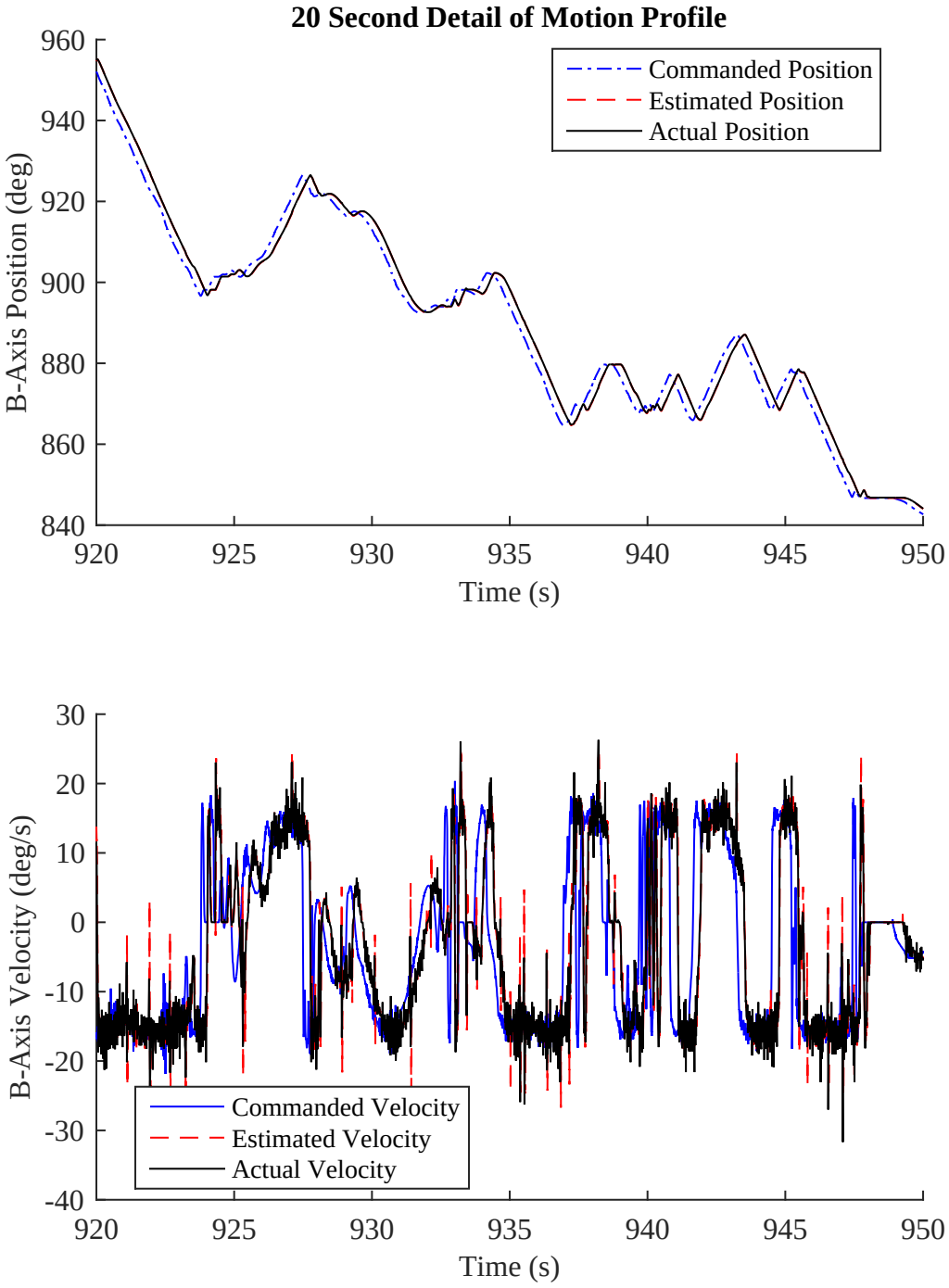


Figure A.20: 20 Second B-axis Position and Velocity Progression for Candleholder Top Toolpath

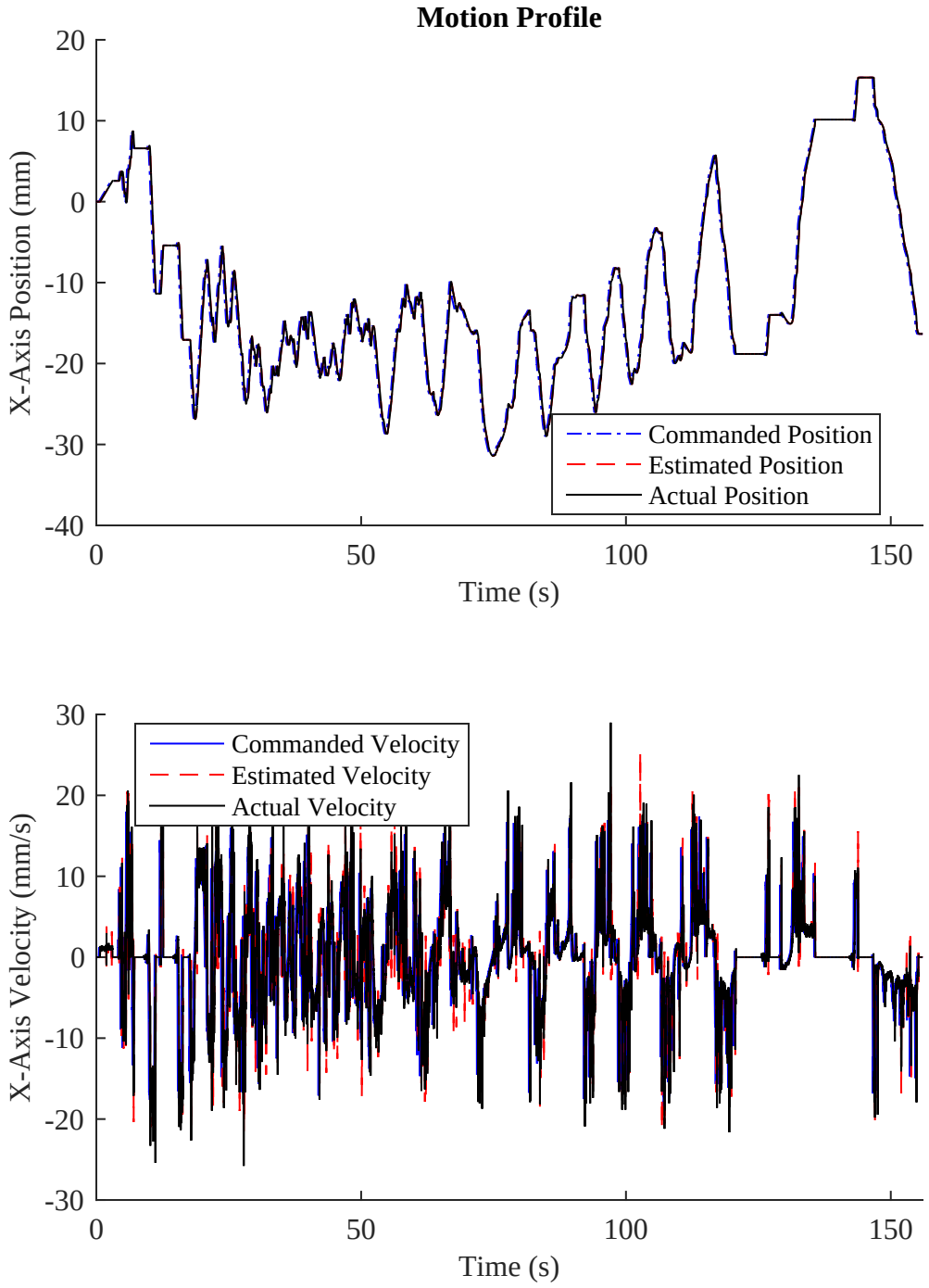


Figure A.21: X-axis Position and Velocity Progression for Candleholder Bottom Toolpath

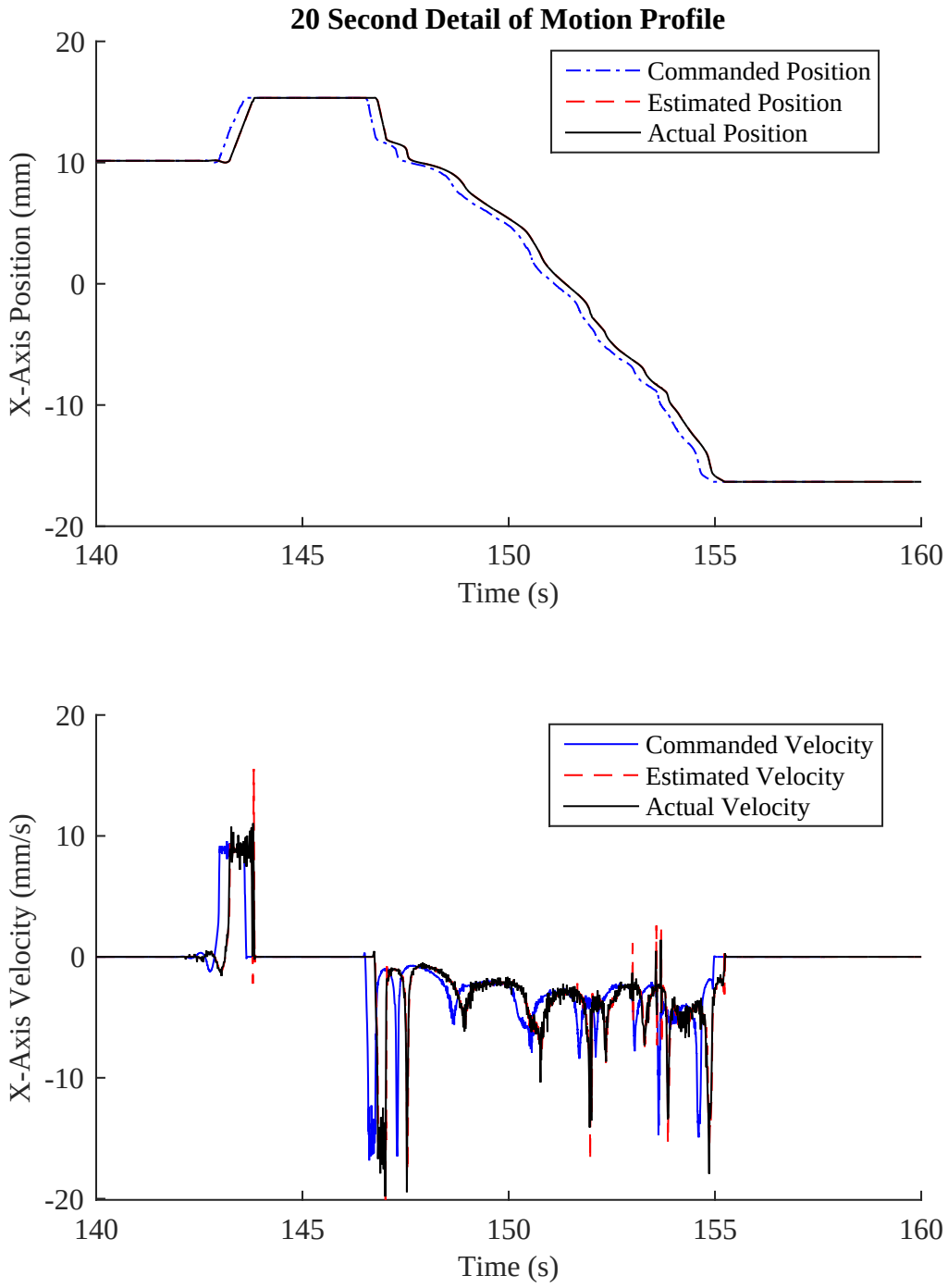


Figure A.22: 20 Second X-axis Position and Velocity Progression for Candleholder Bottom Toolpath

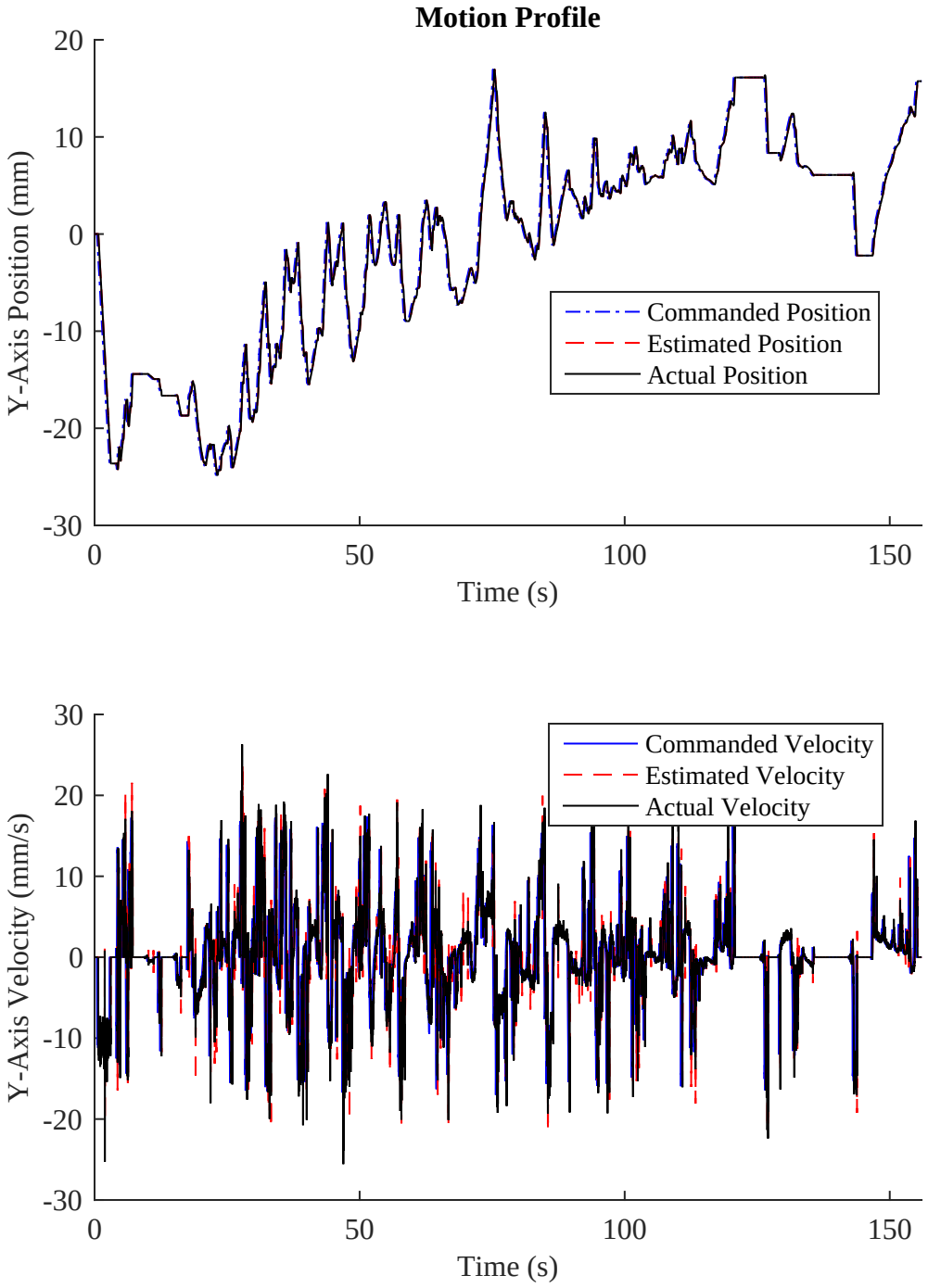


Figure A.23: Y-axis Position and Velocity Progression for Candleholder Bottom Toolpath

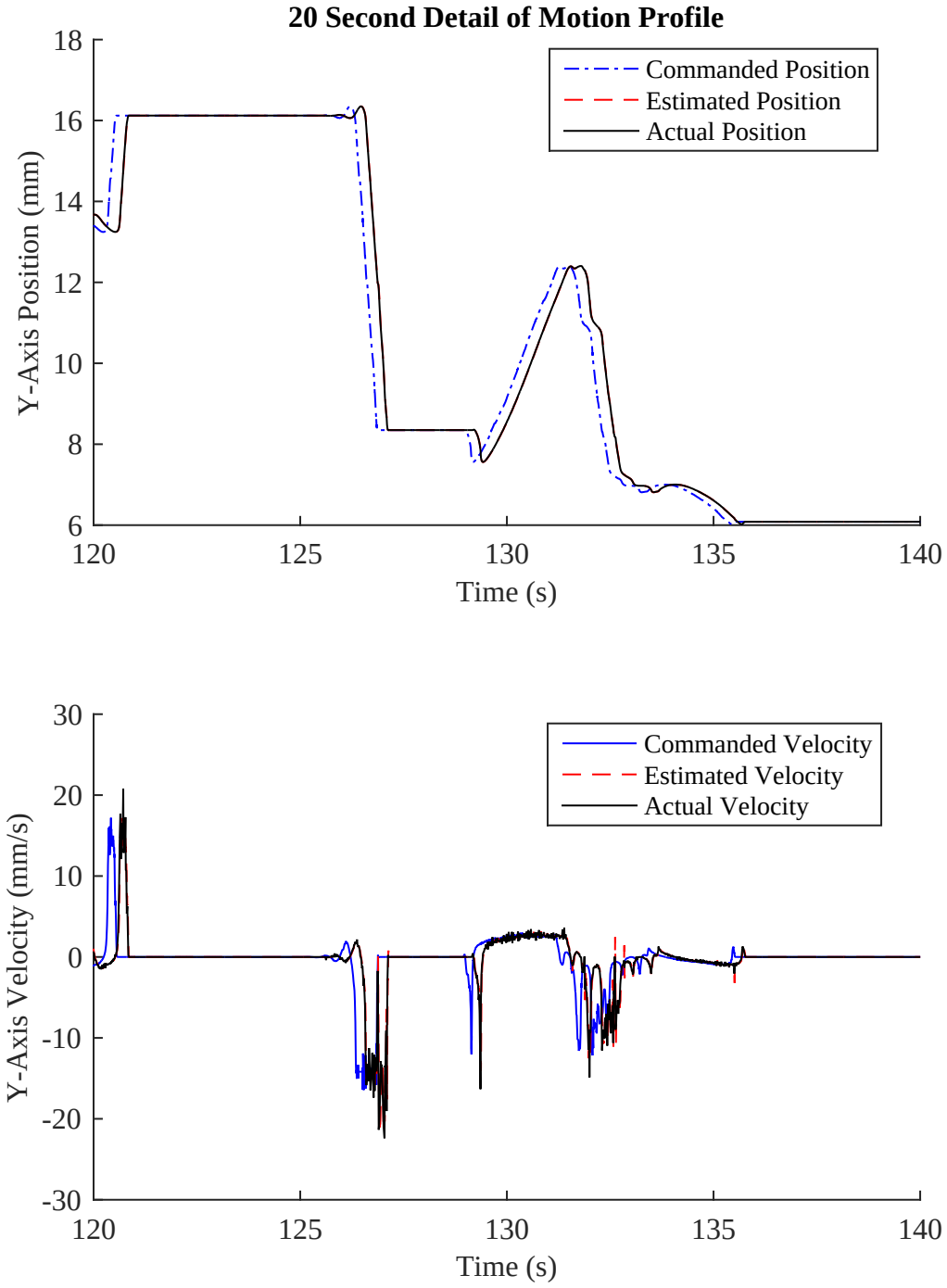


Figure A.24: 20 Second Y-axis Position and Velocity Progression for Candleholder Bottom Toolpath

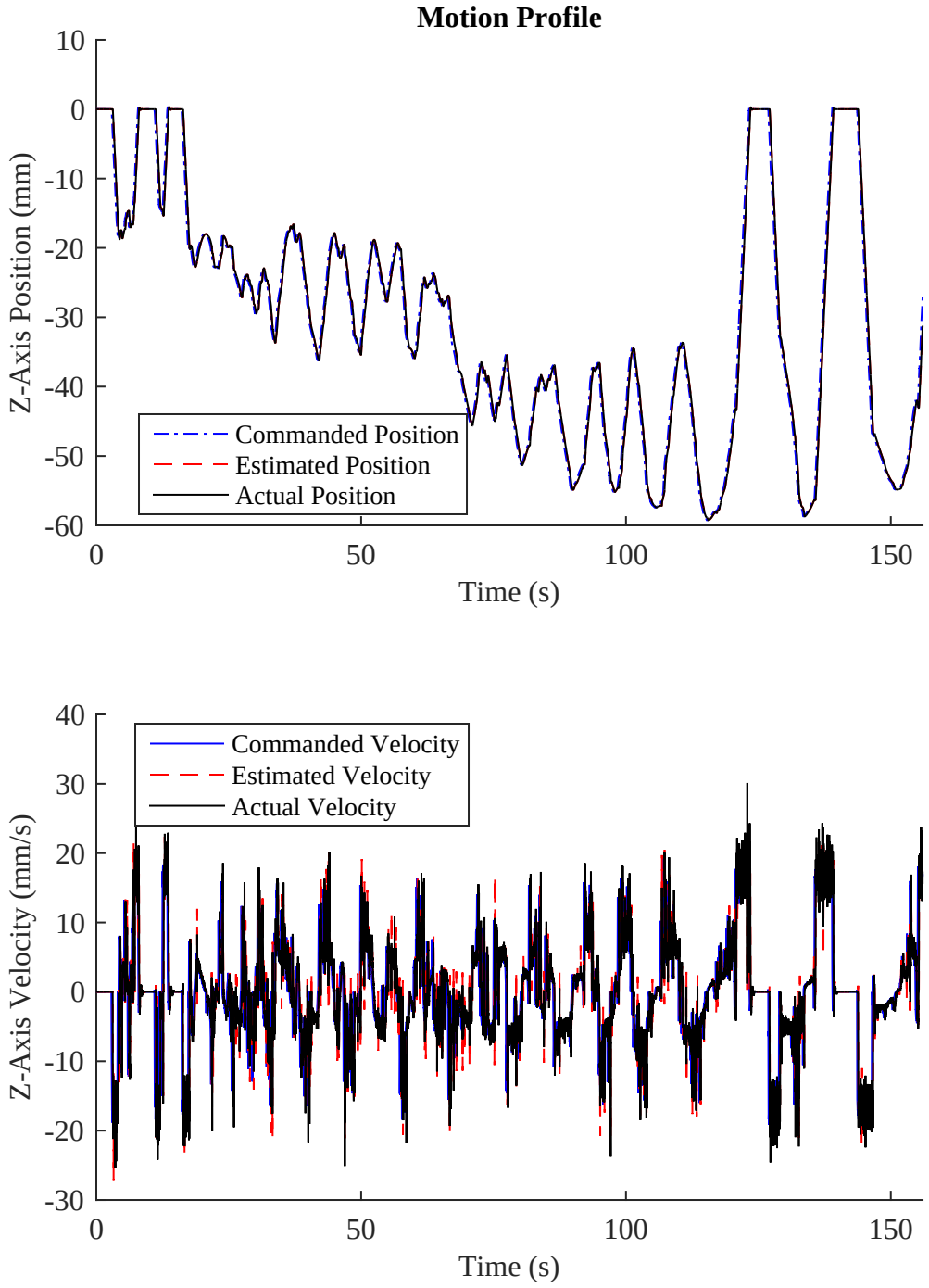


Figure A.25: Z-axis Position and Velocity Progression for Candleholder Bottom Toolpath

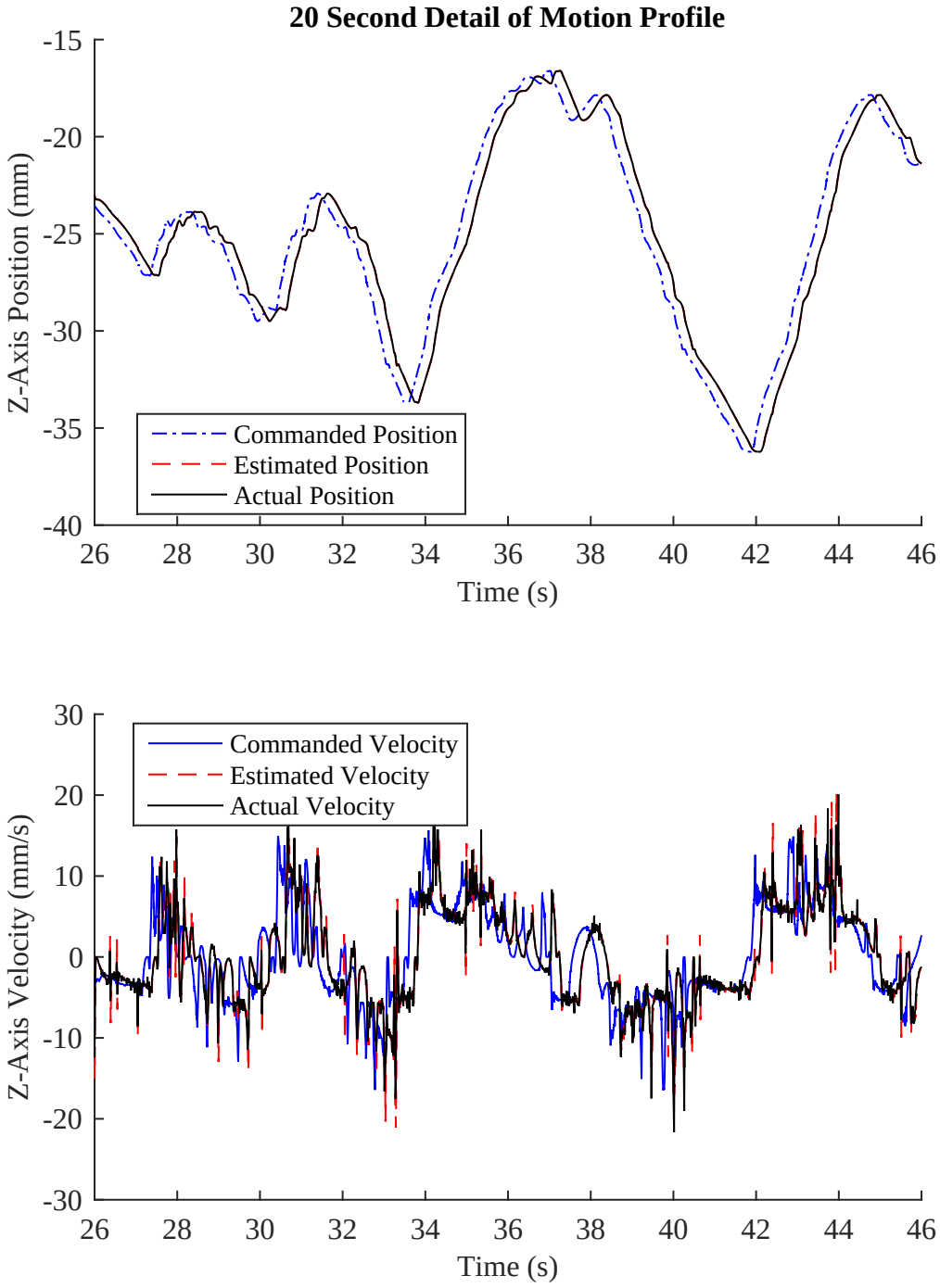


Figure A.26: 20 Second Z-axis Position and Velocity Progression for Candleholder Bottom Toolpath

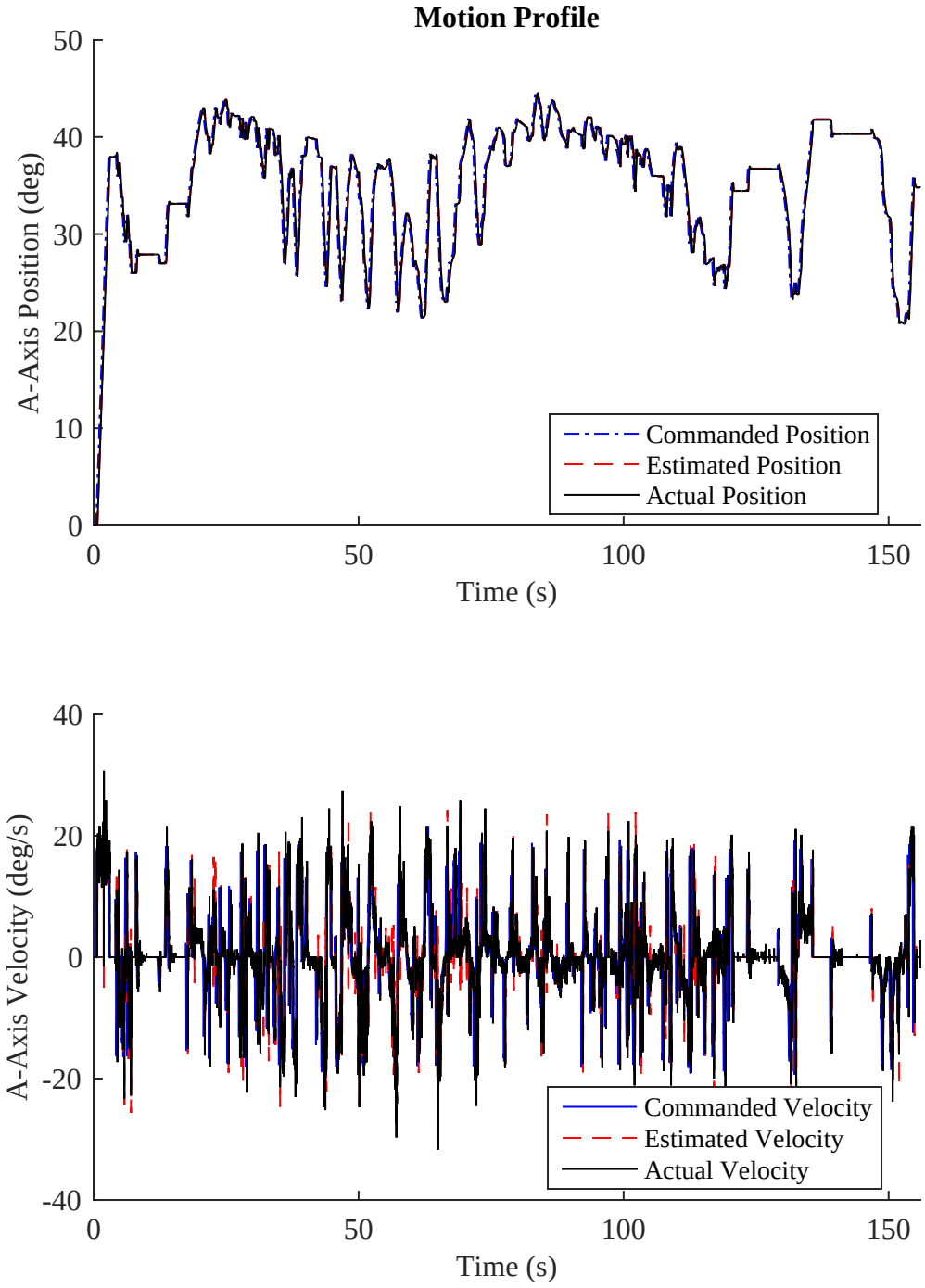


Figure A.27: A-axis Position and Velocity Progression for Candleholder Bottom Toolpath

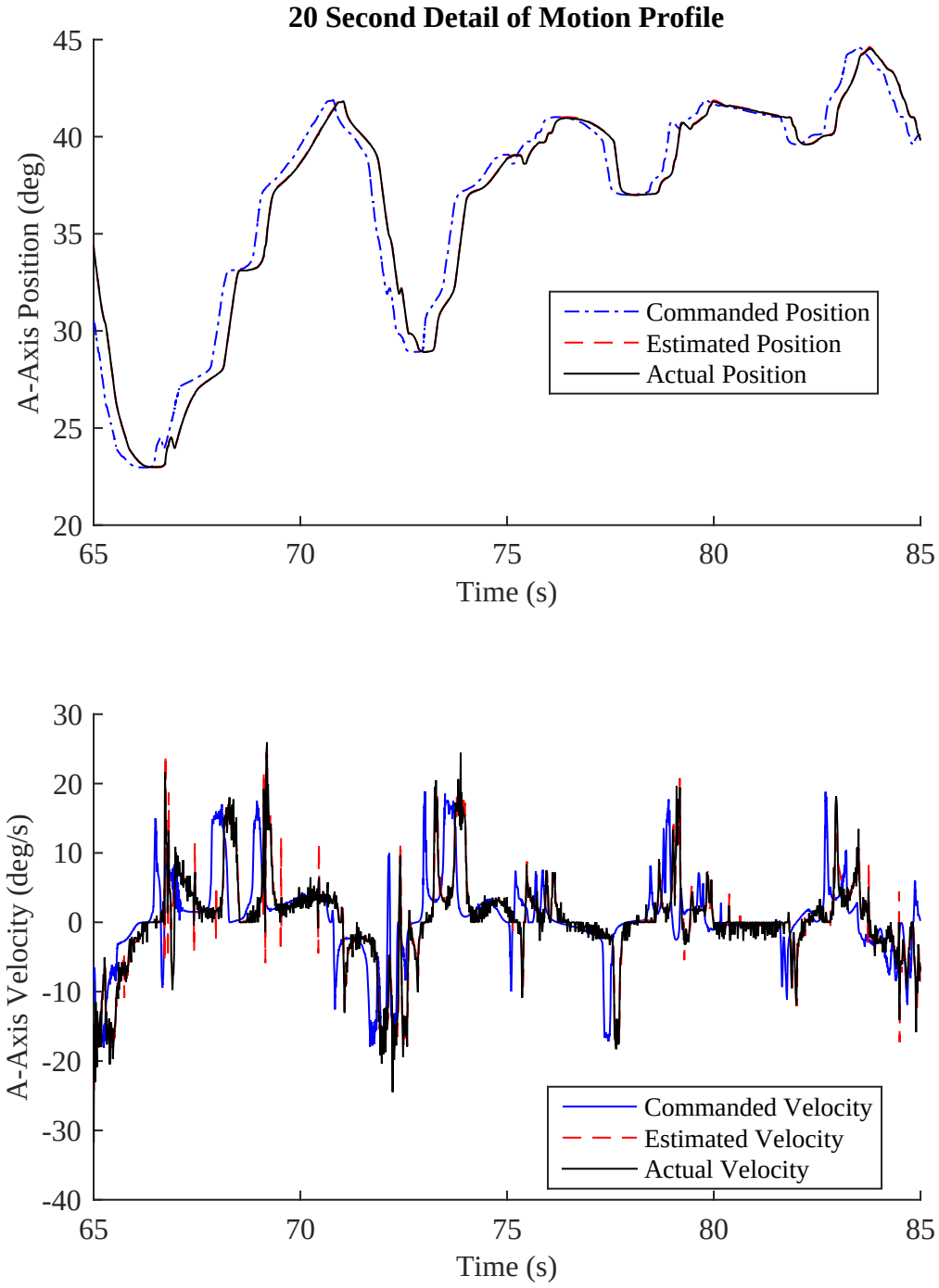


Figure A.28: 20 Second A-axis Position and Velocity Progression for Candleholder Bottom Toolpath

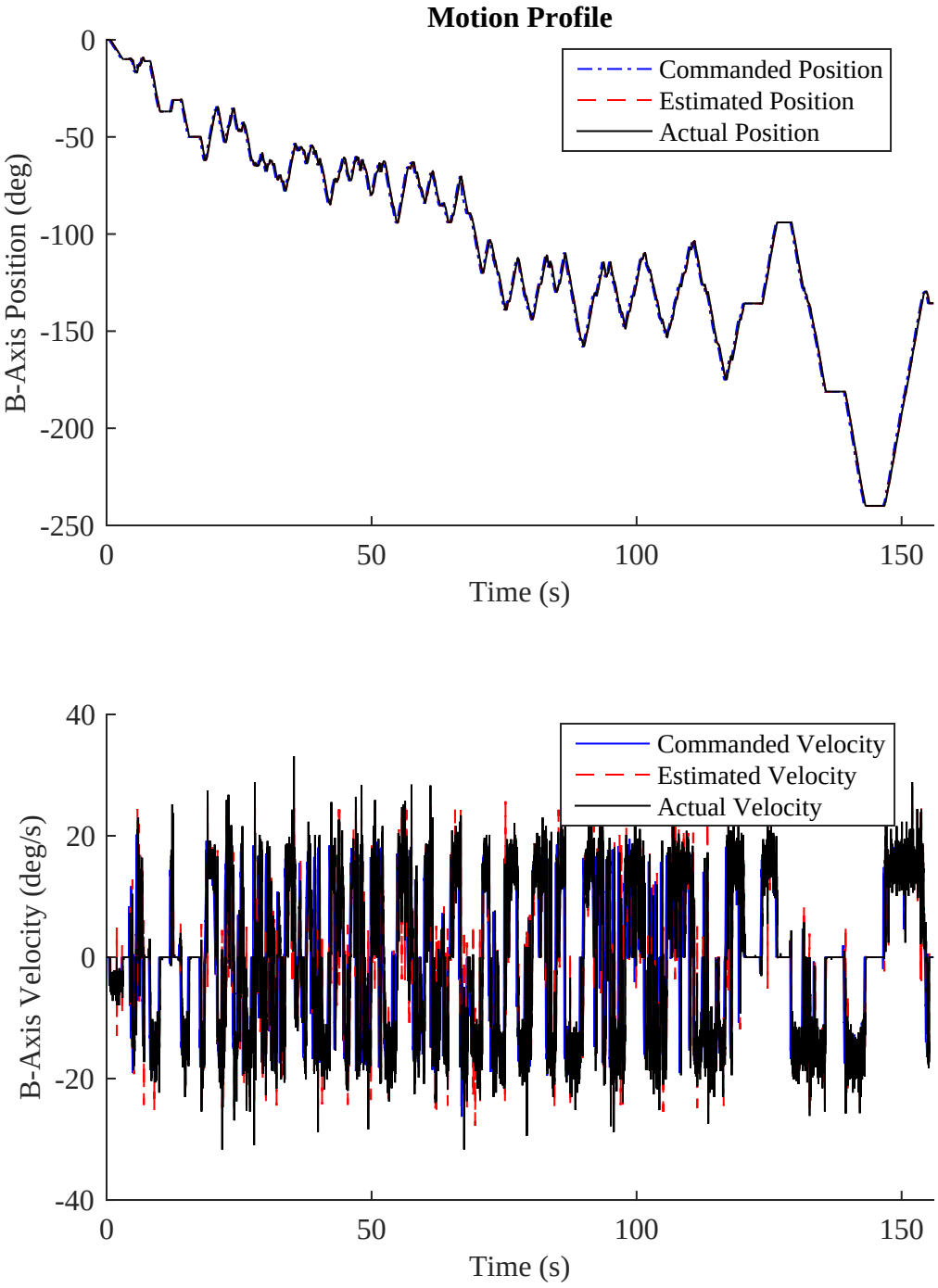


Figure A.29: B-axis Position and Velocity Progression for Candleholder Bottom Toolpath

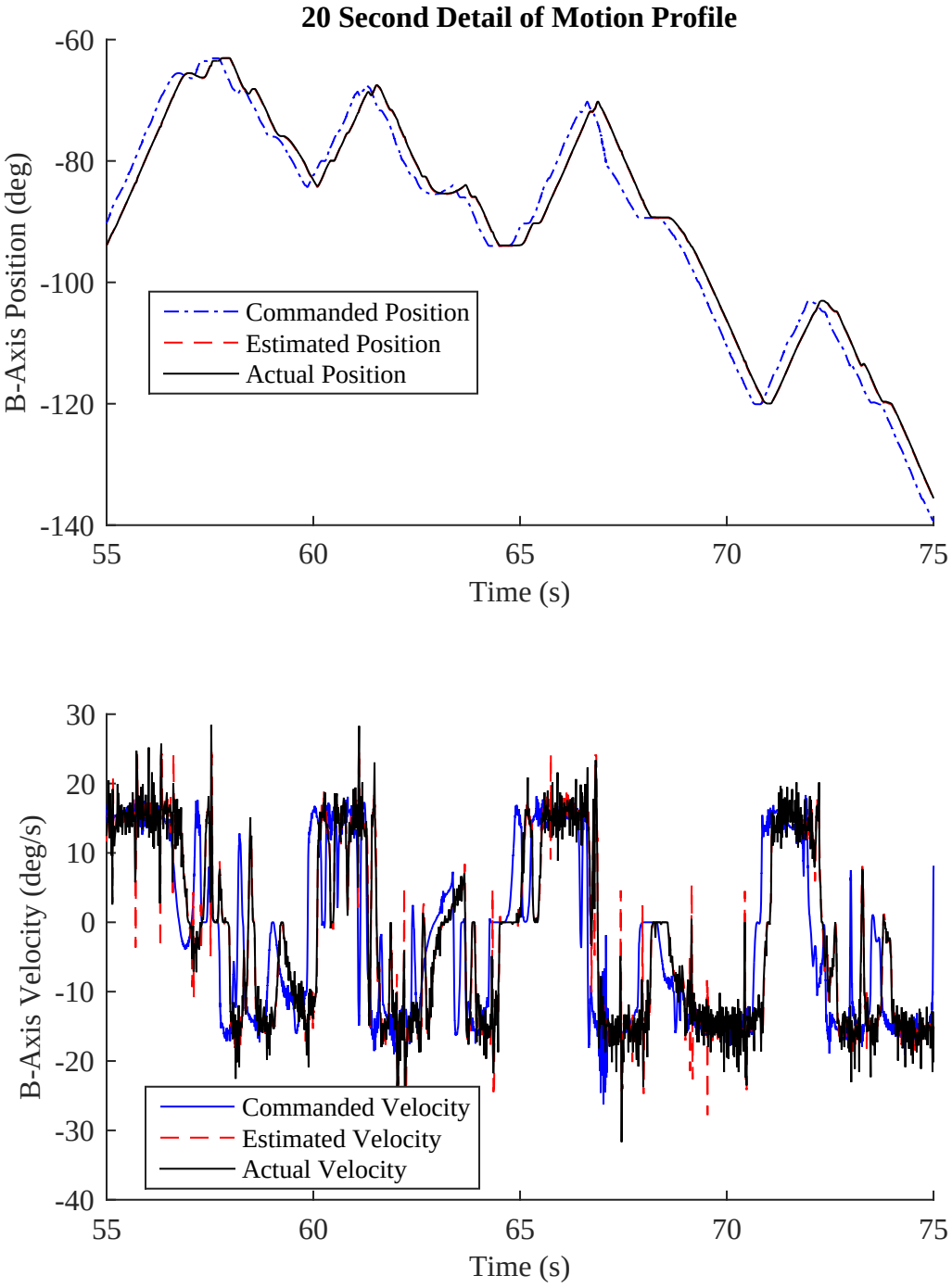


Figure A.30: 20 Second B-axis Position and Velocity Progression for Candleholder Bottom Toolpath

REFERENCES

[1] R. Lynn, M. Dinar, N. Huang, J. Collins, J. Yu, C. Greer, T. Tucker, and T. Kurfess, "Direct Digital Subtractive Manufacturing of a Functional Assembly Using Voxel-Based Models," *Journal of Manufacturing Science and Engineering*, vol. 140, no. 2, p. 021 006, 2017.

[2] International Society of Automation, *ANSI/ISA-95: Enterprise-Control System Integration*, 2010.

[3] T.-C. Chang, R. A. Wysk, and H.-P. Wang, *Computer-aided manufacturing*. Prentice Hall, 1998, p. 748, ISBN: 013754524X.

[4] J. J. Shah and M. Mantyla, *Parametric and feature-based CAD/CAM : concepts, techniques, and applications*. Wiley, 1995, p. 619, ISBN: 0471002143.

[5] L. A. Piegl and W. Tiller, "Computing offsets of NURBS curves and surfaces," *CAD Computer Aided Design*, vol. 31, no. 2, pp. 147–156, 1999.

[6] Z. Yang, L. Y. Shen, C. M. Yuan, and X. S. Gao, "Curve fitting and optimal interpolation for CNC machining under confined error using quadratic B-splines," *CAD Computer Aided Design*, vol. 66, pp. 62–72, 2015.

[7] H. Shen, X. Yao, and J. Fu, "Smooth non-uniform rational B-spline (NURBS) machining with kinematic limit for short linear segments," *International Journal of Computer Integrated Manufacturing*, vol. 24, no. 12, pp. 1103–1116, 2011.

[8] G. J. Jense, "Voxel-based methods for CAD," *Computer-Aided Design*, vol. 21, no. 8, pp. 528–533, 1989.

[9] D. Jang, K. Kim, and J. Jung, "Voxel-based virtual multi-axis machining," *International Journal of Advanced Manufacturing Technology*, vol. 16, no. 10, pp. 709–713, 2000.

[10] Tucker Innovations Inc, *SculptPrint - The Subtractive 3D Printing Application*.

[11] J. A. Tarbutton, T. R. Kurfess, T. Tucker, and D. Konobrytskyi, "Gouge-free Voxel-Based Machining for Parallel Processors," *The International Journal of Advanced Manufacturing Technology*, vol. 69, no. 9, pp. 1941–1953, 2013.

[12] D. Konobrytskyi, "Automated CNC Tool Path Planning and Machining Simulation on Highly Parallel Computing Architectures," PhD thesis, 2013.

[13] J. A. Tarbutton, "Automated Digital Machining for Parallel Processors," PhD thesis, 2011.

[14] R. Lynn, D. Contis, M. M. Hossain, N. Huang, T. M. Tucker, and T. R. Kurfess, "Voxel Model Surface Offsetting for Computer-Aided Manufacturing using Virtualized High-Performance Computing," *Journal Of Manufacturing Systems*, vol. 43, pp. 296–304, 2016.

[15] M. M. Hossain, C. Nath, T. M. Tucker, R. W. Vuduc, and T. R. Kurfess, "A Graphical Approach for Freeform Surface Offsetting with GPU Acceleration for Subtractive 3D Printing," in *11th Manufacturing Science and Engineering Conference (MSEC)*, Blacksburg, Virginia, 2016, V002T04A031.

[16] D. Konobrytskyi, M. M. Hossain, T. M. Tucker, A Joshua, T. R. Kurfess, D. Konobrytskyi, M. M. Hossain, T. M. Tucker, A Joshua, D. Konobrytskyi, and J. A. Tarbutton, "5-Axis tool path planning based on highly parallel discrete volumetric geometry representation : Part I contact point generation," *Computer Aided Design and Applications*, vol. 4360, 2018.

[17] C. Tournier and E. Duc, "Iso-scallop tool path generation in 5-axis milling," *International Journal of Advanced Manufacturing Technology*, vol. 25, no. 9-10, pp. 867–875, 2005.

[18] R. T. Farouki and S. Li, "Optimal tool orientation control for 5-axis CNC milling with ball-end cutters," *Computer Aided Geometric Design*, vol. 30, no. 2, pp. 226–239, 2013.

[19] T. Ye, C. Xiong, Y. Xiong, and C. Zhao, "Kinematics constrained five-axis tool path planning for high material removal rate," *Science China Technological Sciences*, vol. 54, no. 12, pp. 3155–3165, 2011.

[20] J. A. Nemes, S. Asamoah-Attiah, E. Budak, and L. Kops, "Cutting load capacity of end mills with complex geometry," *CIRP Annals - Manufacturing Technology*, vol. 50, no. 1, pp. 65–68, 2001.

[21] L. J. Yeh and T. S. Lan, "The optimal control of material removal rate with fixed tool life and speed limitation," *Journal of Materials Processing Technology*, vol. 121, no. 2-3, pp. 238–242, 2002.

[22] A. Tekeli and E. Budak, "Maximization of chatter-free material removal rate in end milling using analytical methods," *Machining Science and Technology*, vol. 9, no. 2, pp. 147–167, 2005.

[23] L. Wang, "Machine availability monitoring and machining process planning towards Cloud manufacturing," *CIRP Journal of Manufacturing Science and Technology*, vol. 6, no. 4, pp. 263–273, 2013.

[24] B. Buckholtz and L. Wang, "Cloud Manufacturing : Current Trends and Future Implementations," *ASME journal of manufacturing science and engineering*, vol. 137, pp. 1–46, 2015.

[25] X. Xu, L. Wang, and S. T. Newman, "Computer-aided process planning - A critical review of recent developments and future trends," *International Journal of Computer Integrated Manufacturing*, vol. 24, no. 1, pp. 1–31, 2011.

[26] D. Wu, D. W. Rosen, L. Wang, and D. Schaefer, "Cloud-based design and manufacturing: A new paradigm in digital manufacturing and design innovation," *Computer-Aided Design*, vol. 59, pp. 1–14, 2015.

[27] R. Lynn, D. Contis, M. Hossain, N. Huang, T. Tucker, and T. Kurfess, "Extending access to HPC manufacturability feedback software through hardware-accelerated virtualized workstations," in *International Symposium on Flexible Automation, ISFA 2016*, 2016, pp. 271–277, ISBN: 9781509034673.

[28] R. Lynn, K. W. Jablokow, N. Reddy, C. Saldana, T. Tucker, T. W. Simpson, T. Kurfess, and C. Williams, "Using Rapid Manufacturability Analysis Tools to Enhance Design-for-Manufacturing Training in Engineering Education," in *ASME 2016 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference (IDETC/CIE 2016)*, Charlotte, NC, 2016.

[29] R. Lynn, C. Saldana, T. Kurfess, S. N. R. Kantareddy, T. Simpson, K. Jablokow, T. Tucker, S. Tedia, and C. Williams, "Toward Rapid Manufacturability Analysis Tools for Engineering Design Education," *Procedia Manufacturing*, vol. 5, no. 44th SME North American Manufacturing Research Conference (NAMRC 44), pp. 1183–1196, 2016.

[30] M. Dinar, R. Lynn, E. Barnett, A. García, G. Kurfess, T. Tucker, and T. Kurfess, "Easy sculpting: A pilot study in teaching CNC machining to students from disparate backgrounds without learning G-code," *International Journal of Engineering Education*, vol. 33, no. 6, pp. 1868–1877, 2017.

[31] R. Lynn, K. Jablokow, C. Saldana, T. Tucker, and T. Kurfess, "Enhancing undergraduate understanding of subtractive manufacturability through virtualized simulation of CNC machining," in *ASEE Annual Conference and Exposition, Conference Proceedings*, vol. 2017-June, 2017.

[32] R. Lynn, "Enhancing Awareness of Additive and Subtractive Manufacturability with Voxel-Based Simlations," Master's Thesis, Georgia Institute of Technology, 2017.

[33] Electronic Industries Association, *EIA Standard EIA-274-D: Interchangeable variable block data format for positioning, contouring, and contouring/positioning numerically controlled machines*, 1979.

[34] ISO, *ISO6983: Automation systems and integration – Numerical control of machines – Program format and definitions of address words*, 2009.

[35] W. Liu, J.-W. Zhang, S.-M. Zhu, C.-C. Zhang, and T.-J. Yuan, "Efficient tool posture global collision-free area generation for 5-axis point clouds machining," *The International Journal of Advanced Manufacturing Technology*, 2016.

[36] S. Lavernhe, C. Tournier, and C. Lartigue, "Kinematical Performances in 5-Axis Machining," *Advances in Integrated Design and Manufacturing in Mechanical Engineering*, pp. 489–503, 2006.

[37] K. Morishige, K. Kase, and Y. Takeuchi, "Tool path generation using C-space for 5-axis control machining," *Seimitsu Kogaku Kaishi/Journal of the Japan Society for Precision Engineering*, vol. 62, no. 12, pp. 1783–1787, 1996.

[38] R. T. Farouki, C. Y. Han, and S. Li, "Inverse kinematics for optimal tool orientation control in 5-axis CNC machining," *Computer Aided Geometric Design*, vol. 31, no. 1, pp. 13–26, 2014.

[39] Methods Machine Tools Inc., *Understanding 5-axis Kinematics*.

[40] S. Lavernhe, C. Tournier, and C. Lartigue, "Optimization of 5-axis high-speed machining using a surface based approach," *Computer-Aided Design*, vol. 40, no. 10-11, pp. 1015–1023, 2008.

[41] Okuma Corporation, *OSP-P300S Programming Manual*, 2013.

[42] Mazak Corporation, *Mazatrol SmoothX EIA/ISO Programming Manual*, 2015.

[43] T. R. Kramer, F. M. Proctor, and E. Messina, "The NIST RS274NGC Interpreter -Version 3," Tech. Rep., 2000, p. 121.

[44] X. Xu and S. Newman, "Making CNC machine tools more open, interoperable and intelligenta review of the technologies," *Computers in Industry*, vol. 57, no. 2, pp. 141–152, 2006.

[45] X. Xu, *Integrating advanced computer-aided design, manufacturing, and numerical control: principles and implementations*, 11. IGI Global, 2011, vol. 49, pp. 3425–3426, ISBN: 9781599047140.

[46] M. Tolouei-Rad, "Efficient CNC Milling by Adjusting Material Removal Rate," *International Journal of Mechanical and Mechatronics Engineering*, vol. 5, no. 10, pp. 342–346, 2011.

[47] B. Denkena and E. Hasselberg, "Influence of the cutting tool compliance on the workpiece surface shape in face milling of workpiece compounds," in *Procedia CIRP*, vol. 31, 2015, pp. 7–12.

[48] ISO, *ISO 10303-238:2007. Industrial automation systems and integration – Product data representation and exchange – Part 238: Application protocol: Application interpreted model for computerized numerical controllers*, 2007.

[49] ——, *ISO 14649: Industrial Automation Systems and Integration - Physical Device Control - Data Model for Computerized Numerical Controllers*, 2003.

[50] X. W. Xu, "Realization of STEP-NC enabled machining," *Robotics and Computer-Integrated Manufacturing*, vol. 22, no. 2, pp. 144–153, 2006.

[51] M. Hardwick, "On STEP-NC and the Complexities of Product Data Integration," *Journal of Computing and Information Science in Engineering*, vol. 4, no. 1, p. 60, 2004.

[52] S. N. Grigoriev and G. M. Martinov, "The Control Platform for Decomposition and Synthesis of Specialized CNC Systems," *Procedia CIRP*, vol. 2, no. 3, pp. 858–863, 2015.

[53] S. Mitsui, F. Tanaka, and T. Kishinami, "Machining Feature-Driven 5-Axis CNC Machine Tools," in *Towards Synthesis of Micro-/Nano-systems*, Springer London, 2007, pp. 169–174, ISBN: 978-1-84628-559-2.

[54] H. Liang and X. Li, "Five-axis STEP-NC controller for machining of surfaces," *International Journal of Advanced Manufacturing Technology*, vol. 68, no. 9-12, pp. 2791–2800, 2013.

[55] M Hardwick and D Loffredo, "Lessons learned implementing STEP-NC AP-238," *International Journal of Computer Integrated Manufacturing*, vol. 19, no. 6, pp. 523–532, 2006.

[56] R. B. Jerard and O. Ryou, "NCML: a data exchange format for internet-based machining," *International Journal of Computer Applications in Technology*, vol. 26, no. 1/2, p. 75, 2006.

[57] X. W. Xu, H. Wang, J. Mao, S. T. Newman, T. R. Kramer, F. M. Proctor, and J. L. Michaloski, "STEP-compliant NC research: the search for intelligent CAD/CAPP/CAM/CNC integration," *International Journal of Production Research*, vol. 43, no. 17, pp. 3703–3743, 2005.

[58] H. Kagermann, W. Wahlster, and J. Helbig, *Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0*, 2013.

[59] W. Sobel, "MTConnect Standard Part 1 - Overview and Protocol," *The Asociation for Manufacturing Technology*, pp. 0–70, 2012.

[60] S. Singh, A. Angrish, J. Barkley, B. Starly, Y. S. Lee, and P. Cohen, "Streaming Machine Generated Data to Enable a Third-Party Ecosystem of Digital Manufacturing Apps," *Procedia Manufacturing*, vol. 10, pp. 1020–1030, 2017.

[61] G. Ćwikła, "Methods of Manufacturing Data Acquisition for Production Management - A Review," *Advanced Materials Research*, vol. 837, pp. 618–623, 2013.

[62] B. E. Lee, J. Michaloski, F. Proctor, S. Venkatesh, and N. Bengtsson, "MTConnect-Based Kaizen for Machine Tool Processes," in *Volume 3: 30th Computers and Information in Engineering Conference, Parts A and B*, ASME, 2010, pp. 1183–1190, ISBN: 978-0-7918-4411-3.

[63] B. Edrington, B. Zhao, A. Hansel, M. Mori, and M. Fujishima, "Machine monitoring system based on MTConnect technology," in *Procedia CIRP*, vol. 22, Elsevier, 2014, pp. 92–97.

[64] F. Ridwan and X. Xu, "Advanced CNC system with in-process feed-rate optimisation," *Robotics and Computer-Integrated Manufacturing*, vol. 29, no. 3, pp. 12–20, 2013.

[65] X. F. Liu, S. M.N. A. Sunny, M. R. Shahriar, M. C. Leu, M. Cheng, and L. Hu, "Implementation of MTConnect for Open Source 3D Printers in Cyber Physical Manufacturing Cloud," in *Volume 1A: 36th Computers and Information in Engineering Conference*, ASME, 2016, ISBN: 978-0-7918-5007-7.

[66] R. Lynn, E. Wescoat, D. Han, and T. Kurfess, "Embedded Fog Computing for High-Frequency MTConnect Data Analytics," *Manufacturing Letters*, 2017.

[67] R. Lynn, A. Chen, S. Locks, C. Nath, and T. Kurfess, "Intelligent and Accessible Data Flow Architectures for Manufacturing System Optimization," in *International Conference on Advances in Production Management Systems (APMS): Innovative Production Management Towards Sustainable Growth*, Tokyo, Japan, 2015.

[68] M. Parto, M. Dinar, and T. Kurfess, "An MTConnect-Compatible Platform for Secured Machine Monitoring through Integration of Fog Computing, Cloud Computing, and Communication Protocols," in *Proceedings of the 2018 International Symposium on Flexible Automation (ISFA 2018)*, 2018.

[69] R. Lynn, W. Louhichi, M. Parto, E. Wescoat, and T. Kurfess, "Rapidly Deployable MTConnect-Based Machine Tool Monitoring Systems," *MSEC 2017*, pp. 1–10, 2017.

[70] P. D. Urbina Coronado, R. Lynn, W. Louhichi, M. Parto, E. Wescoat, and T. Kurfess, *Part data integration in the Shop Floor Digital Twin: Mobile and cloud technologies to enable a manufacturing execution system*, 2018.

[71] M. Helu, A. Joseph, and T. Hedberg, "A standards-based approach for linking as-planned to as-fabricated product data," *CIRP Annals*, vol. 67, no. 1, pp. 487–490, 2018.

[72] M. Hardwick, "Operate, Orchestrate, and Originate," Tech. Rep., 2017.

[73] J. Michaloski and B. Lee, "Quantifying the performance of mt-connect in a distributed manufacturing environment," in *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, vol. 2, 2009, pp. 1–7, ISBN: 9780791848999.

[74] M. Helu and B. Weiss, "The Current State of Sensing, Health Management, and Control for Small-to-Medium-Sized Manufacturers," in *Proceedings of the 2016 Manufacturing Science and Engineering Conference (MSEC 2016)*, Blacksburg, Virginia: ASME, 2016, ISBN: 978-0-7918-4990-3.

[75] R. Lynn, W. Louhichi, M. Parto, E. Wescoat, and T. Kurfess, "Rapidly Deployable MTConnect-Based Machine Tool Monitoring Systems," in *2017 ASME Manufacturing Science and Engineering Conference (MSEC)*, 2017.

[76] R. Lynn, A. Chen, S. Locks, C. Nath, and T. Kurfess, "Intelligent and Accessible Data Flow Architectures for Manufacturing System Optimization," in *IFIP Advances in Information and Communication Technology*, vol. 459, 2015, pp. 27–35, ISBN: 9783319227559.

[77] OPC Foundation, *OPC UA Specification: Part 1 - Overview and Concepts*, 2017.

[78] M. Schleipen, "OPC UA supporting the automated engineering of production monitoring and control systems," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, IEEE, 2008, pp. 640–647, ISBN: 1424415063.

[79] T. Hannelius, M. Salmenperä, and S. Kuikka, "Roadmap to adopting OPC UA," in *IEEE International Conference on Industrial Informatics (INDIN)*, IEEE, 2008, pp. 756–761, ISBN: 9781424421718.

[80] S.-H. Leitner and W. Mahnke, "OPC UA Service-oriented Architecture for Industrial Applications," *Softwaretechnik-Trends*, vol. 26, no. 4, pp. 1 –6, 2006.

[81] A. Balador, N. Ericsson, and Z. Bakhshi, "Communication middleware technologies for industrial distributed control systems: A literature review," in *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA*, IEEE, 2018, pp. 1–6, ISBN: 9781509065059.

[82] D. Treffer, P. Wahl, D. Markl, G. Koscher, E. Roblegg, and J. G. Khinast, "Hot Melt Extrusion as a Continuous Pharmaceutical Manufacturing Process," in *Melt Extrusion: Materials, Technology and Drug Product Design*, Springer, New York, NY, 2013, pp. 363–396, ISBN: 9781461484325.

[83] T. Terzimehic, M. Wenger, A. Zoitl, A. Bayha, K. Becker, T. Muller, and H. Schauerte, "Towards an industry 4.0 compliant control software architecture using IEC 61499 & OPC UA," in *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE, 2017, pp. 1–4, ISBN: 978-1-5090-6505-9.

[84] S. Rohjans, M. Uslar, and H. J. Appelrath, "OPC UA and CIM: Semantics for the smart grid," in *2010 IEEE PES Transmission and Distribution Conference and Exposition: Smart Solutions for a Changing World*, IEEE, 2010, pp. 1–8, ISBN: 9781424465477.

[85] M. Hoffmann, T. Meisen, and S. Jeschke, "OPC UA based ERP agents: Enabling scalable communication solutions in heterogeneous automation environments," in *International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS)*, Springer, Cham, 2017, pp. 120–131, ISBN: 9783319599298.

[86] M. Hoffmann, C. Büscher, T. Meisen, and S. Jeschke, "Continuous Integration of Field Level Production Data into Top-level Information Systems Using the OPC Interface Standard," in *Procedia CIRP*, vol. 41, Elsevier, 2016, pp. 496–501.

[87] D. Mourtzis, N. Milas, and A. Vlachou, "An Internet of Things-Based Monitoring System for Shop-Floor Control," *Journal of Computing and Information Science in Engineering*, vol. 18, no. 2, 2018.

[88] F. Tao, J. Cheng, and Q. Qi, "IIHub: An industrial internet-of-things hub toward smart manufacturing based on cyber-physical system," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 5, pp. 2271–2280, 2018.

[89] A. Girbea, C. Suciu, S. Nechifor, and F. Sisak, "Design and implementation of a service-oriented architecture for the optimization of industrial applications," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 1, pp. 185–196, 2014.

[90] D. Wu, S. Liu, L. Zhang, J. Terpenny, R. X. Gao, T. Kurfess, and J. A. Guzzo, "A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing," *Journal of Manufacturing Systems*, vol. 43, pp. 25–34, 2017.

[91] L. E. Oliveira and A. J. Álvares, "Axiomatic Design Applied to the Development of a System for Monitoring and Teleoperation of a CNC Machine through the Internet," in *Procedia CIRP*, vol. 53, Elsevier, 2016, pp. 198–205, ISBN: 5506181772668.

[92] N. M. Torrisi and J. F. G. De Oliveira, "Remote monitoring for high-speed CNC processes over public IP networks using CyberOPC," *International Journal of Advanced Manufacturing Technology*, vol. 60, no. 1-4, pp. 191–200, 2012.

[93] J. Albus and R. Lumia, "The Enhanced Machine Controller (EMC): An Open Architecture Controller for Machine Tools," *Journal of Manufacturing Review*, vol. Vol. 7, no. No. 3, pp. 278–280, 1994.

[94] K. Adams and S. Huang, "RTX: A Real-Time Operating System Environment for CNC Machine Tool Control," *IFAC Proceedings Volumes*, vol. 25, no. 7, pp. 55–60, 1992.

[95] R. L. Hecker, G. M. Flores, Q. Xie, and R. Haran, "Servocontrol of machine-tools: a review," *Latin American Applied Research*, vol. 38, no. 1, pp. 85–94, 2008.

[96] C. Zhang, C. Shan, and H. Wang, "DESIGN AND REALIZATION OF G CODE INTERPRETER FOR CNC SYSTEM," *Journal of Shandong University of Technology*, 2002.

[97] Y. Altintas and W. K. Munasinghe, "A Hierarchical Open-Architecture CNC System for Machine Tools," *CIRP Annals - Manufacturing Technology*, vol. 43, no. 1, pp. 349–354, 1994.

[98] D Renton and M. A. Elbestawi, "Motion control for linear motor feed drives in advanced machine tools," *International Journal of Machine Tools and Manufacture*, vol. 41, no. 4, pp. 479–507, 2001.

[99] A. A. Agrachev, G. Stefani, and P. Zezza, "Strong Optimality for a Bang-Bang Trajectory," *SIAM Journal on Control and Optimization*, vol. 41, no. 4, pp. 991–1014, 2002.

[100] H. M. Schaettler, "ON THE TIME-OPTIMALITY OF BANG-BANG TRAJECTORIES IN R3," *BULLETIN (New Series) OF THE AMERICAN MATHEMATICAL SOCIETY*, vol. 16, no. 1, 1987.

[101] C. S. Wilson, "Precision Contouring Error Analysis," in *American Society for Precision Engineering Spring Topical Meeting on Mechanisms and Control for Ultra-Precision Motion*, 1994.

[102] S. Li, Q. Zhang, X. Gao, and H. Li, "Minimum time trajectory planning for five-axis machining with general kinematic constraints," *Mathematics Mechanization Research Preprints*, vol. 31, pp. 1–20, 2012.

[103] S. H. Nam and M. Y. Yang, "A study on a generalized parametric interpolator with real-time jerk-limited acceleration," *CAD Computer Aided Design*, vol. 36, no. 1, pp. 27–36, 2004.

[104] M. S. Tsai, H. W. Nien, and H. T. Yau, "Development of an integrated look-ahead dynamics-based NURBS interpolator for high precision machinery," *CAD Computer Aided Design*, vol. 40, no. 5, pp. 554–566, 2008.

[105] M. M. Emami and B. Arezoo, "A look-ahead command generator with control over trajectory and chord error for NURBS curve with unknown arc length," *CAD Computer Aided Design*, vol. 42, no. 7, pp. 625–632, 2010.

[106] K. Erkorkmaz and Y. Altintas, "High speed CNC system design. Part I: Jerk limited trajectory generation and quintic spline interpolation," *International Journal of Machine Tools and Manufacture*, vol. 41, no. 9, pp. 1323–1345, 2001.

[107] B. Sencer, Y. Altintas, and E. Croft, "Feed optimization for five-axis CNC machine tools with drive constraints," *International Journal of Machine Tools and Manufacture*, vol. 48, no. 7-8, pp. 733–745, 2008.

[108] R. T. Farouki, J. Manjunathaiah, D. Nicholas, G. F. Yuan, and S. Jee, "Variable-feedrate CNC interpolators for constant material removal rates along Pythagorean-hodograph curves," *CAD Computer Aided Design*, vol. 30, no. 8, pp. 631–640, 1998.

[109] T. M. Jahns, "Motion Control with Permanent-Magnet AC Machines," *Proceedings of the IEEE*, vol. 82, no. 8, pp. 1241–1252, 1994.

[110] M Minhat, V Vyatkin, X Xu, S Wong, and Z Al-Bayaa, "A novel open CNC architecture based on STEP-NC data model and IEC 61499 function blocks," *Robotics and Computer-Integrated Manufacturing*, vol. 25, no. 3, pp. 560–569, 2009. arXiv: arXiv:1011.1669v3.

[111] Y. F. Tsai, R. T. Farouki, and B. Feldman, "Performance analysis of CNC interpolators for time-dependent feedrates along PH curves," *Computer Aided Geometric Design*, vol. 18, no. 3, pp. 245–265, 2001.

[112] X. Beudaert, S. Lavernhe, and C. Tournier, "Direct trajectory interpolation on the surface using an open CNC," *International Journal of Advanced Manufacturing Technology*, vol. 75, no. 1-4, pp. 535–546, 2014.

[113] M. Y. Cheng, M. C. Tsai, and J. C. Kuo, "Real-time NURBS command generators for CNC servo controllers," *International Journal of Machine Tools and Manufacture*, vol. 42, no. 7, pp. 801–813, 2002.

[114] G. T. Chiu and M. Tomizuka, "Contouring control of machine tool feed drive systems: A task coordinate frame approach," *IEEE Transactions on Control Systems Technology*, vol. 9, no. 1, pp. 130–139, 2001.

[115] X. Li, H. Zhao, X. Zhao, and H. Ding, "Dual sliding mode contouring control with high accuracy contour error estimation for five-axis CNC machine tools," *International Journal of Machine Tools and Manufacture*, vol. 108, pp. 74–82, 2016.

[116] J. E. Bobrow, S. Dubowsky, and J. S. Gibson, "Time-Optimal Control of Robotic Manipulators Along Specified Paths," *The International Journal of Robotics Research*, vol. 4, no. 3, pp. 3–17, 1985.

[117] H. Pham and Q. C. Pham, "A New Approach to Time-Optimal Path Parameterization Based on Reachability Analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018. arXiv: 1707.07239.

[118] The Xenomai Project, *Xenomai Realtime System*.

[119] IEEE Computer Society, *754-1985 - IEEE Standard for Binary Floating-Point Arithmetic*, 1985.

[120] Python 3.5 Documentation, *17.2. multiprocessing: Process-based parallelism*.

[121] ——, *Global Interpreter Lock*.